NetCDF User's Guide

An Interface for Data Access
Version 2.4
February 1996

Russ Rew, Glenn Davis, Steve Emmerson, and Harvey Davies
Unidata Program Center

Foreword

Unidata is a National Science Foundation-sponsored program empowering U.S. universities, through innovative applications of computers and networks, to make the best use of atmospheric and related data for enhancing education and research. For analyzing and displaying such data, the Unidata Program Center offers universities several supported software packages developed by other organizations, including the University of Wisconsin, Purdue University, NASA, and the National Weather Service. Underlying these is a Unidata-developed system for acquiring and managing data in real time, making practical the Unidata principle that each university should acquire and manage its own data holdings as local requirements dictate. It is significant that the Unidata program has no data center — the management of data is a “distributed” function.

The Network Common Data Form (netCDF) software described in this guide was originally intended to provide a common data access method for the various Unidata applications. These deal with a variety of data types that encompass single-point observations, time series, regularly-spaced grids, and satellite or radar images.

The netCDF software functions as an I/O library, callable from C or FORTRAN, which stores and retrieves data in self-describing, machine-independent files. Each netCDF file can contain an unlimited number of multi-dimensional, named variables (with differing types that include integers, reals, characters, bytes, etc.), and each variable may be accompanied by ancillary data, such as units of measure or descriptive text. The interface includes a method for appending data to existing netCDF files in prescribed ways, functionality that is not unlike a (fixed length) record structure. However, the netCDF library also allows direct-access storage and retrieval of data by variable name and index and therefore is useful only for disk-resident (or memory-resident) files.

NetCDF access has been implemented in about half of Unidata’s software, so far, and it is planned that such commonality will extend across all Unidata applications in order to:

- Facilitate the use of common data files by distinct applications.
- Permit data files to be transported between or shared by dissimilar computers transparently, i.e., without translation.
- Reduce the programming effort usually spent interpreting formats in a way that is equally effective for FORTRAN and C programmers.
- Reduce errors arising from misinterpreting data and ancillary data.
- Facilitate using output from one application as input to another.
- Establish an interface standard which simplifies the inclusion of new software into the Unidata system.

A measure of success has been achieved. NetCDF is now in use on computing platforms that range from CRAYs to Personal Computers and include most UNIX-based workstations. It can be used to create a complex dataset on one computer (say in FORTRAN) and retrieve that same self-describing dataset on another computer (say in C) without intermediate translations — netCDF files can be transferred across a network, or they can be accessed remotely using a suitable network file system.

Because we believe that the use of netCDF access in non-Unidata software will benefit Unidata's primary constituency — such use may result in more options for analyzing and displaying Unidata information — the netCDF library is distributed without licensing or other significant restrictions, and current versions can be obtained via anonymous FTP. Apparently the software has been well received by a wide range of institutions beyond the atmospheric science community, and a substantial number of public domain and commercial data analysis systems can now accept netCDF files as input.

Several organizations have adopted netCDF as a data access standard, and there is an effort underway at the National Center for Supercomputer Applications (NCSA, which is associated with the University of Illinois at Urbana-Champaign) to support the netCDF programming interfaces as a means to store and retrieve data in "HDF files," i.e., in the format used by the popular NCSA tools. We have encouraged and cooperated with these efforts.

Questions occasionally arise about the level of support provided for the netCDF software. Unidata's formal position, stated in the copyright notice which accompanies the netCDF library, is that the software is provided "as is". In practice, the software is updated from time to time, and Unidata intends to continue making improvements for the foreseeable future. Because Unidata's mission is to serve geoscientists at U.S. universities, problems reported by that community necessarily receive the greatest attention.

We hope the reader will find the software useful and will give us feedback on its application as well as suggestions for its improvement.

David Fulker
Unidata Program Center Director
University Corporation for Atmospheric Research

Summary

The purpose of the Network Common Data Form (netCDF) interface is to allow you to create, access, and share array-oriented data in a form that is self-describing and network-transparent. "Self-describing" means that a file includes information defining the data it contains. "Network-transparent" means that a file is represented in a form that can be accessed by computers with different ways of storing integers, characters, and floating-point numbers. Using the netCDF interface for creating new datasets makes the data portable. Using the netCDF interface in software for data access, management, analysis, and display can make the software more generally useful.

The netCDF software includes C and FORTRAN interfaces for accessing netCDF data. These libraries are available for many common computing platforms. C++ and perl interfaces for netCDF data access are also available from Unidata. The community of netCDF users has contributed ports of the software to additional platforms and interfaces for other programming languages as well. Source code for netCDF software libraries is freely available to encourage the sharing of both array-oriented data and the software that makes the data useful.

This User's Guide presents the netCDF data model, but documents only the C and FORTRAN interfaces. Separate documents are available for C++ and perl interfaces. Reference documentation for UNIX systems, in the form of UNIX 'man' pages for the C and FORTRAN interfaces, is available with the netCDF software. Extensive additional information about netCDF, including pointers to other software that works with netCDF data, is available at the netCDF World Wide Web site ('`http://www.unidata.ucar.edu/packages/netcdf/`').

1 Introduction

1.1 The NetCDF Interface

The Network Common Data Form, or netCDF, is an interface to a library of data access functions for storing and retrieving data in the form of arrays. An *array* is an n-dimensional (where n is 0, 1, 2, ...) rectangular structure containing items which all have the same *data type* (e.g. 8-bit character, 32-bit integer). A *scalar* (simple single value) is a 0-dimensional array.

NetCDF is an abstraction that supports a view of data as a collection of self-describing, network-transparent objects that can be accessed through a simple interface. Array values may be accessed directly, without knowing details of how the data are stored. Auxiliary information about the data, such as what units are used, may be stored with the data. Generic utilities and application programs can access netCDF files and transform, combine, analyze, or display specified fields of the data. The development of such applications may lead to improved accessibility of data and improved reusability of software for array-oriented data management, analysis, and display.

The netCDF software implements an *abstract data type*, which means that all operations to access and manipulate data in a netCDF file must use only the set of functions provided by the interface. The representation of the data is hidden from applications that use the interface, so that how the data are stored could be changed without affecting existing programs. The physical representation of netCDF data is designed to be independent of the computer on which the data were written.

Unidata supports the netCDF interfaces for C, FORTRAN, C++, and perl and for various UNIX operating systems. The software is also ported and tested on a few other operating systems, with assistance from users with access to these systems, before each major release. Unidata's netCDF software is freely available via FTP to encourage its widespread use.

1.2 NetCDF is Not a Database Management System

Why not use an existing database management system for storing array-oriented data? Relational database software is not suitable for the kinds of data access supported by the netCDF interface.

First, existing database systems that support the relational model do not support multidimensional objects (arrays) as a basic unit of data access. Representing arrays as relations makes some useful kinds of data access awkward and provides little support for the abstractions of multidimensional data and coordinate systems. A quite different data model is needed for array-oriented data to facilitate its retrieval, modification, mathematical manipulation and visualization.

Related to this is a second problem with general-purpose database systems: their poor performance on large arrays. Collections of satellite images, scientific model outputs and long-term global weather observations are beyond the capabilities of most database systems to organize and index for efficient retrieval.

Finally, general-purpose database systems provide, at significant cost in terms of both resources and access performance, many facilities that are not needed in the analysis, management, and display of array-oriented data. For example, elaborate update facilities, audit trails, report formatting, and mechanisms designed for transaction-processing are unnecessary for most scientific applications.

1.3 File Format

To achieve network-transparency (machine-independence), netCDF is implemented in terms of XDR (eXternal Data Representation, see '`ftp://ds.internic.net/rfc/rfc1832.txt`'), a proposed standard protocol for describing and encoding data. XDR provides encoding of data into machine-independent sequences of bits. XDR has been implemented on a wide variety of computers, by assuming only that eight-bit bytes can be encoded and decoded in a consistent way. XDR uses the IEEE floating-point standard for floating-point data.

The overall structure of netCDF files is described in Chapter 9 [NetCDF File Structure and Performance], page 121.

The details of the format are described in Appendix B [File Format Specification], page 143. However, users are discouraged from using the format specification to develop independent low-level software for reading and writing netCDF files, because this could lead to compatibility problems when the format is modified.

1.4 What about Performance?

One of the goals of netCDF is to support efficient access to small subsets of large datasets. To support this goal, netCDF uses direct access rather than sequential access. This can be much more efficient when data is read in a different order from that in which it was written.

The amount of XDR overhead depends on many factors, including the data type, the type of computer, the granularity of data access, and how well the implementation has been tuned to the computer on which it is run. This overhead is typically small in comparison to the overall resources used by an application. In any case, the overhead of the XDR layer is usually a reasonable price to pay for portable, network-transparent data access.

Although efficiency of data access has been an important concern in designing and implementing netCDF, it is still possible to use the netCDF interface to access data in inefficient ways: for example, by requesting a slice of data that requires a single value from each record. Advice on how to use the interface efficiently is provided in Chapter 9 [NetCDF File Structure and Performance], page 121.

1.5 Is NetCDF a Good Archive Format?

NetCDF can be used as a general-purpose archive format for storing arrays. Compression of data is possible with netCDF (e.g., using arrays of eight-bit or 16-bit integers to encode low-resolution floating-point numbers instead of arrays of 32-bit numbers), but the current version of netCDF was not designed to achieve optimal compression of data. Hence, using netCDF may require more space than special-purpose archive formats that exploit knowledge of particular characteristics of specific datasets.

1.6 Creating Self-Describing Data conforming to Conventions

The mere use of netCDF is not sufficient to make data "self-describing" and meaningful to both humans and machines. The names of variables and dimensions should be meaningful and conform to any relevant conventions. Dimensions should have corresponding coordinate variables where sensible.

Attributes play a vital role in providing ancillary information. It is important to use all the relevant standard attributes using the relevant conventions. Section 8.1 [Attribute Conventions], page 101, describes reserved attributes (used by the netCDF library) and attribute conventions for generic application software.

A number of groups have defined their own additional conventions and styles for netCDF data. Descriptions of these conventions, as well as examples incorporating them can be accessed from the netCDF Conventions site (`http://www.unidata.ucar.edu/packages/netcdf/conventions.html`).

These conventions should be used where suitable. Additional conventions are often needed for local use. These should be contributed to the above netCDF Conventions site if likely to interest other users in similar areas.

1.7 Background and Evolution of the NetCDF Interface

The development of the netCDF interface began with a modest goal related to Unidata's needs: to provide a common interface between Unidata applications and ingested real-time meteorological data. Since Unidata software was intended to run on multiple hardware platforms with access from both C and FORTRAN, achieving Unidata's goals had the potential for providing a package that was useful in a broader context. By making the package widely available and collaborating with other organizations with similar needs, we hoped to improve the then current situation in which software for scientific data access was only rarely reused by others in the same discipline and almost never reused between disciplines (Fulker, 1988).

Important concepts employed in the netCDF software originated in a paper (Treinish and Gough, 1987) that described data-access software developed at the NASA Goddard National Space Science Data Center (NSSDC). The interface provided by this software was called the Common Data

Format (CDF). The NASA CDF was originally developed as a platform-specific FORTRAN library to support an abstraction for storing arrays.

The NASA CDF package had been used for many different kinds of data in an extensive collection of applications. It had the virtues of simplicity (only 13 subroutines), independence from storage format, generality, ability to support logical user views of data, and support for generic applications.

Unidata held a workshop on CDF in Boulder in August 1987. We proposed exploring the possibility of collaborating with NASA to extend the CDF FORTRAN interface, to define a C interface, and to permit the access of data aggregates with a single call, while maintaining compatibility with the existing NASA interface.

Independently, Dave Raymond at the New Mexico Institute of Mining and Technology had developed a package of C software for UNIX that supported sequential access to self-describing array-oriented data and a "pipes and filters" (or "data flow") approach to processing, analyzing, and displaying the data. This package also used the "Common Data Format" name, later changed to C-Based Analysis and Display System (CANDIS). Unidata learned of Raymond's work (Raymond, 1988), and incorporated some of his ideas, such as the use of named dimensions and variables with differing shapes in a single data object, into the Unidata netCDF interface.

In early 1988, Glenn Davis of Unidata developed a prototype netCDF package in C that was layered on XDR. This prototype proved that a single-file, network-transparent implementation of the CDF interface could be achieved at acceptable cost and that the resulting programs could be implemented on both UNIX and VMS systems. However, it also demonstrated that providing a small, portable, and NASA CDF-compatible FORTRAN interface with the desired generality was not practical. NASA's CDF and Unidata's netCDF have since evolved separately, but recent CDF versions share many characteristics with netCDF.

In early 1988, Joe Fahle of SeaSpace, Inc. (a commercial software development firm in San Diego, California), a participant in the 1987 Unidata CDF workshop, independently developed a CDF package in C that extended the NASA CDF interface in several important ways (Fahle, 1989). Like Raymond's package, the SeaSpace CDF software permitted variables with unrelated shapes to be included in the same data object and permitted a general form of access to multidimensional arrays. Fahle's implementation was used at SeaSpace as the intermediate form of storage for a variety of steps in their image-processing system. This interface and format have subsequently evolved into the Terascan data format.

After studying Fahle's interface, we concluded that it solved many of the problems we had identified in trying to stretch the NASA interface to our purposes. In August 1988, we convened a small workshop to agree on a Unidata netCDF interface, and to resolve remaining open issues. Attending were Joe Fahle of SeaSpace, Michael Gough of Apple (an author of the NASA CDF software), Angel Li of the University of Miami (who had implemented our prototype netCDF software on VMS and was a potential user), and Unidata systems development staff. Consensus was reached at the workshop after some further simplifications were discovered. A document incorporating the results of the workshop into a proposed Unidata netCDF interface specification was distributed widely for comments before Glenn Davis and Russ Rew implemented the first version of the software. Comparison with other data-access interfaces and experience using netCDF are discussed in (Rew and Davis, 1990a), (Rew and Davis, 1990b), (Jenter and Signell, 1992), and (Brown, Folk, Goucher, and Rew, 1993).

In October 1991, we announced version 2.0 of the netCDF software distribution. Slight modifications to the C interface (declaring dimension sizes to be `long` rather than `int`) improved the usability of netCDF on inexpensive platforms such as MS-DOS computers, without requiring recompilation on other platforms. This change to the interface required no changes to the associated file format.

Release of netCDF version 2.3.2 in June 1993 preserved the same file format but added single call access to records, optimizations for accessing cross-sections involving non-contiguous data, subsampling along specified dimensions (using 'strides'), accessing non-contiguous data (using 'mapped array sections'), improvements to the ncdump and ncgen utilities, and an experimental C++ interface.

1.8 What's New Since the Previous Release?

This Guide documents the February 1996 release of netCDF 2.4, which preserves the same file format as earlier versions but includes the following changes from version 2.3.2:

- support for new platforms;
- significant Cray optimizations;
- improved ease of installation;
- revised documentation;
- additions to the C++ interface; and
- fixes for reported bugs.

In order to support netCDF on new platforms where the size of a `long` is greater than the size of an `int`, the new release fully integrates the use of the `nclong` typedef into the C and C++ interfaces.

Additions and changes were made to the C++ interface to make it easier to step through records, coordinate concurrent access to netCDF files, and access single records.

1.9 Limitations of NetCDF

The netCDF data model is widely applicable to data that can be organized into a collection of named array variables with named attributes, but there are some important limitations to the model and its implementation in software.

The data model does not support nested data structures. The netCDF interface provides little help in representing trees, nested arrays, or other recursive data structures, mostly because of the requirement that the FORTRAN interface should be able to read and write any netCDF dataset. Through use of indirection and conventions it is possible to represent some kinds of nested structures, but the result falls short of the netCDF goal of "self-describing data".

A significant limitation of the current implementation is that only one unlimited dimension is permitted for each netCDF dataset. Multiple variables can share an unlimited dimension, but then they must all grow together. Hence the netCDF model does not cater for variables with several changeable dimension sizes. It is also not possible to have different changeable dimensions in different variables within the same file. Variables that have non-rectangular shapes (e.g. "ragged arrays") cannot be represented conveniently.

The interface does not provide any facilities specific to coordinate variables, such as a using them to specify position along dimensions as an alternative to normal indexing. There are no facilities yet for packing data in bit fields (XDR lacks this capability). Hence an array of 9-bit data must be stored in 16-bit arrays to be conveniently accessed. Dataset sizes are currently limited to 2 Gigabytes, because of the use of 32-bit signed offsets.

Finally, the current implementation limits concurrent access to a netCDF file. One writer and multiple readers may access data in a single file simultaneously, but there is no support for multiple concurrent writers.

1.10 Future Plans for NetCDF

XDR is to be replaced by new software under development. This will provide added functionality and greater efficiency.

Current plans are to add transparent data packing, improved concurrency support, access to data by key or coordinate value, support for efficient structure changes (e.g. new variables and attributes), new data types, and the addition of type-safe C and FORTRAN interfaces for accessing data as a specific type, independent of how it is stored. Other desirable extensions that may be added, if practical, include support for pointers to data cross-sections in other files, nested arrays (allowing representation of ragged arrays, trees and other recursive data structures), ability to access datasets larger than 2 Gigabytes, and multiple unlimited dimensions.

References

1. Brown, S. A, M. Folk, G. Goucher, and R. Rew, "Software for Portable Scientific Data Management," *Computers in Physics*, American Institute of Physics, Vol. 7, No. 3, May/June 1993.

2. Fahle, J., *TeraScan Applications Programming Interface*, SeaSpace, San Diego, California, 1989.

3. Fulker, D. W., "The netCDF: Self-Describing, Portable Files—a Basis for 'Plug-Compatible' Software Modules Connectable by Networks," *ICSU Workshop on Geophysical Informatics*, Moscow, USSR, August 1988.
4. Fulker, D. W., "Unidata Strawman for Storing Earth-Referencing Data," *Seventh International Conference on Interactive Information and Processing Systems for Meteorology, Oceanography, and Hydrology*, New Orleans, La., American Meteorology Society, January 1991.
5. Gough, M. L., *NSSDC CDF Implementer's Guide (DEC VAX/VMS) Version 1.1*, National Space Science Data Center, 88-17, NASA/Goddard Space Flight Center, 1988.
6. Jenter, H. L. and R. P. Signell, "NetCDF: A Freely-Available Software-Solution to Data-Access Problems for Numerical Modelers," *Proceedings of the American Society of Civil Engineers Conference on Estuarine and Coastal Modeling*, Tampa, Florida, 1992.
7. Raymond, D. J., "A C Language-Based Modular System for Analyzing and Displaying Gridded Numerical Data," *Journal of Atmospheric and Oceanic Technology*, **5**, 501-511, 1988.
8. Rew, R. K. and G. P. Davis, "The Unidata netCDF: Software for Scientific Data Access," *Sixth International Conference on Interactive Information and Processing Systems for Meteorology, Oceanography, and Hydrology*, Anaheim, California, American Meteorology Society, February 1990.
9. Rew, R. K. and G. P. Davis, "NetCDF: An Interface for Scientific Data Access," *Computer Graphics and Applications*, IEEE, pp. 76-82, July 1990.
10. Treinish, L. A. and M. L. Gough, "A Software Package for the Data Independent Management of Multi-Dimensional Data," *EOS Transactions*, American Geophysical Union, **68**, 633-635, 1987.

2 Components of a NetCDF File

2.1 The NetCDF Data Model

A netCDF file contains *dimensions*, *variables*, and *attributes*, which all have both a name and an ID number by which they are identified. These components can be used together to capture the meaning of data and relations among data fields in an array-oriented dataset. The netCDF library allows simultaneous access to multiple netCDF files which are identified by file ID numbers, in addition to ordinary file names.

A netCDF file contains a symbol table for variables containing their name, data type, rank (number of dimensions), dimensions, and starting disk address. Each element is stored at a disk address which is a linear function of the array indices (subscripts) by which it is identified. This obviates the need for these indices to be stored, either as fields within records, or in an index to the records (as in a relational database). This provides a fast and compact storage method, unless there are many missing values.

2.1.1 Naming Conventions

The names of dimensions, variables and attributes consist of arbitrary sequences of alphanumeric characters (as well as underscore '_' and hyphen '-'), beginning with a letter or underscore. (However names commencing with underscore are reserved for system use.) Case is significant in netCDF names.

2.1.2 network Common Data Form Language (CDL)

We will use a small netCDF example to illustrate the concepts of the netCDF data model. This includes dimensions, variables, and attributes. The notation used to describe this simple netCDF object is called CDL (network Common Data form Language), which provides a convenient way of describing netCDF files. The netCDF system includes utilities for producing human-oriented CDL text files from binary netCDF files and vice versa.

```
netcdf example_1 {  // example of CDL notation for a netCDF file

dimensions:          // dimension names and sizes are declared first
        lat = 5, lon = 10, level = 4, time = unlimited;

variables:           // variable types, names, shapes, attributes
        float   temp(time,level,lat,lon);
                    temp:long_name     = "temperature";
                    temp:units         = "celsius";
        float   rh(time,lat,lon);
                    rh:long_name = "relative humidity";
                    rh:valid_range = 0.0, 1.0;      // min and max
        int     lat(lat), lon(lon), level(level);
                    lat:units       = "degrees_north";
                    lon:units       = "degrees_east";
                    level:units     = "millibars";
        short   time(time);
                    time:units      = "hours since 1996-1-1";
        // global attributes
                    :source = "Fictional Model Output";

data:                // optional data assignments
        level   = 1000, 850, 700, 500;
        lat     = 20, 30, 40, 50, 60;
        lon     = -160,-140,-118,-96,-84,-52,-45,-35,-25,-15;
        time    = 12;
        rh      =.5,.2,.4,.2,.3,.2,.4,.5,.6,.7,
                 .1,.3,.1,.1,.1,.1,.5,.7,.8,.8,
                 .1,.2,.2,.2,.2,.5,.7,.8,.9,.9,
                 .1,.2,.3,.3,.3,.3,.7,.8,.9,.9,
                  0,.1,.2,.4,.4,.4,.4,.7,.9,.9;
}
```

The CDL notation for a netCDF file can be generated automatically by using `ncdump`, a utility program described later (see Section 10.5 [ncdump], page 130). Another netCDF utility, `ncgen`, generates a netCDF file (or optionally C or FORTRAN source code containing calls needed to produce a netCDF file) from CDL input (see Section 10.4 [ncgen], page 129).

The CDL notation is simple and largely self-explanatory. It will be explained more fully as we describe the components of a netCDF file. For now, note that CDL statements are terminated by a semicolon. Spaces, tabs, and newlines can be used freely for readability. Comments in CDL follow the characters '`//`' on any line. A CDL description of a netCDF file takes the form

```
netCDF name {
  dimensions: ...
  variables: ...
  data: ...
}
```

where the *name* is used only as a default in constructing file names by the `ncgen` utility. The CDL description consists of three optional parts, introduced by the keywords `dimensions`, `variables`, and `data`. NetCDF dimension declarations appear after the `dimensions` keyword, netCDF variables and attributes are defined after the `variables` keyword, and variable data assignments appear after the `data` keyword.

2.2 Dimensions

A dimension may be used to represent a real physical dimension, for example, time, latitude, longitude, or height. A dimension might also be used to index other quantities, for example station or model-run-number.

A netCDF dimension has both a *name* and a *size*. A dimension size is an arbitrary positive integer, except that one dimension in a netCDF file can have the size `UNLIMITED`.

Such a dimension is called the *unlimited dimension* or the *record dimension*. A variable with an unlimited dimension can grow to any length along that dimension. The unlimited dimension index is like a record number in conventional record-oriented files. A netCDF file can have at most one unlimited dimension, but need not have any. If a variable has an unlimited dimension, that dimension must be the most significant (slowest changing) one. Thus any unlimited dimension must be the first dimension in a CDL shape (and first in C declarations, but last in FORTRAN).

CDL dimension declarations may appear on one or more lines following the CDL keyword `dimensions`. Multiple dimension declarations on the same line may be separated by commas. Each declaration is of the form *name* = *size*.

There are four dimensions in the above example: `lat`, `lon`, `level`, and `time`. The first three are assigned fixed sizes; `time` is assigned the size `UNLIMITED`, which means it is the *unlimited* dimension.

The basic unit of named data in a netCDF file is a *variable*. When a variable is defined, its *shape* is specified as a list of dimensions. These dimensions must already exist.

The number of dimensions is called the *rank* (a.k.a. *dimensionality*). A scalar variable has rank 0, a vector has rank 1 and a matrix has rank 2.

It is possible to use the same dimension more than once in specifying a variable shape. For example, `correlation(instrument, instrument)` could be a correlation matrix giving correlations between measurements using different instruments. But data whose dimensions correspond to those of physical space/time should have a shape comprising different dimensions, even if some of these have the same size.

2.3 Variables

Variables are used to store the bulk of the data in a netCDF file. A *variable* represents an array of values of the same type. A scalar value is treated as a 0-dimensional array. A variable has a name, a data type, and a shape described by its list of dimensions specified when the variable is created. A variable may also have associated attributes, which may be added, deleted or changed after the variable is created.

A variable data type is one of a small set of netCDF *types* that have the names `NC_BYTE`, `NC_CHAR`, `NC_SHORT`, `NC_LONG`, `NC_FLOAT`, and `NC_DOUBLE` in the C interface and the corresponding names `NCBYTE`, `NCCHAR`, `NCSHORT`, `NCLONG`, `NCFLOAT`, and `NCDOUBLE` in the FORTRAN interface. In the CDL notation, these types are given the simpler names `byte`, `char`, `short`, `long`, `float`, and `double`. `int` may be used as a synonym for `long` and `real` may be used as a synonym for `float` in the CDL notation. We will postpone a discussion of the exact meaning of each of the types until Section 3.1 [NetCDF Data Types], page 21.

CDL variable declarations appear after the `variables` keyword in a CDL unit. They have the form

type variable_name (*dim_name_1, dim_name_2, . . .*) ;

for variables with dimensions, or

type variable_name ;

for scalar variables.

In the above CDL example there are six variables. As discussed below, four of these are coordinate variables. The remaining variables (sometimes called *primary variables*), `temp` and `rh`, contain what is usually thought of as the data. Each of these variables has the unlimited dimension `time` as its first dimension, so they are called *record variables*. A variable that is not a record variable has a fixed size (number of data values) given by the product of its dimension sizes. The size of a record variable is also the product of its dimension sizes, but in this case the product is variable because it involves the size of the unlimited dimension, which can vary. The size of the unlimited dimension is the number of records.

2.3.1 Coordinate Variables

It is legal for a variable to have the same name as a dimension. Such variables have no special meaning to the netCDF library. However there is a convention that such variables should be treated in a special way by software using this library.

A variable with the same name as a dimension is called a *coordinate variable*. It typically defines a physical coordinate corresponding to that dimension. The above CDL example includes the coordinate variables `lat`, `lon`, `level` and `time`, defined as follows:

```
            int     lat(lat), lon(lon), level(level);
            short   time(time);
       ...
    data:
            level   = 1000, 850, 700, 500;
            lat     = 20, 30, 40, 50, 60;
```

```
lon     = -160,-140,-118,-96,-84,-52,-45,-35,-25,-15;
time    = 12;
```

These define the latitudes, longitudes, barometric pressures and times corresponding to positions along these dimensions. Thus there is data at altitudes corresponding to 1000, 850, 700 and 500 millibars; and at latitudes 20, 30, 40, 50 and 60 degrees north. Note that each coordinate variable is a vector and has a shape consisting of just the dimension with the same name.

A position along a dimension can be specified using an *index*. This is an integer with a minimum value of 0 for C programs and 1 for FORTRAN. Thus the 700 millibar level would have an index value of 2 for C and 3 for FORTRAN.

If a dimension has a corresponding coordinate variable, then this provides an alternative, and often more convenient, means of specifying position along it. Current application packages that make use of coordinate variables commonly assume they are numeric vectors and strictly monotonic (all values are different and either increasing or decreasing). There are plans to define more general conventions to allow such things as text labels as values of coordinate variables.

2.4 Attributes

NetCDF *attributes* are used to store data about the data (*ancillary data* or *metadata*), similar in many ways to the information stored in data dictionaries and schema in conventional database systems. Most attributes provide information about a specific variable. These are identified by the name (or ID) of that variable, together with the name of the attribute.

Some attributes provide information about the file as a whole and are called *global* attributes. These are identified by the attribute name together with a blank variable name (in CDL) or a special null variable ID (in C or Fortran).

An attribute has an associated variable (null for a global attribute), a name, a data type, a length, and a value. The current version treats all attributes as vectors; scalar values are treated as single-element vectors.

Conventional attribute names should be used where applicable. New names should be as meaningful as possible.

The type of an attribute is specified when it is created. The types permitted for attributes are the same as the netCDF data types for variables. Attributes with the same name for different variables should sometimes be of different types. For example, the attribute `valid_max` specifying the maximum valid data value for a variable of type `long` should be of type `long`, whereas the attribute `valid_max` for a variable of type `double` should instead be of type `double`.

Attributes are more dynamic than variables or dimensions; they can be deleted and have their type, length, and values changed after they are created, whereas the netCDF interface provides no way to delete a variable or to change its type or shape.

The CDL notation for defining an attribute is

> *variable_name:attribute_name* = *list_of_values* ;

for a variable attribute, or

> *:attribute_name* = *list_of_values* ;

for a global attribute. The type and length of each attribute are not explicitly declared in CDL; they are derived from the values assigned to the attribute. All values of an attribute must be of the same type. The notation used for constant values of the various netCDF types is discussed later (see Section 10.3 [CDL Notation for Data Constants], page 128).

In the netCDF example (see Chapter 2 [CDL example], page 13), `units` is an attribute for the variable `lat` that has a 13-character array value '`degrees_north`'. And `valid_range` is an attribute for the variable `rh` that has length 2 and values '`0.0`' and '`1.0`'.

One global attribute—`source`—is defined for the example netCDF file. This is a character array intended for documenting the data. Actual netCDF files might have more global attributes to document the origin, history, conventions, and other characteristics of the file as a whole.

Most generic applications that process netCDF files assume standard attribute conventions and it is strongly recommended that these be followed unless there are good reasons for not doing so. See Section 8.1 [Attribute Conventions], page 101, for information about `units`, `long_name`, `valid_min`, `valid_max`, `valid_range`, `scale_factor`, `add_offset`, `_FillValue`, and other conventional attributes.

Attributes may be added to a netCDF file long after it is first defined, so you don't have to anticipate all potentially useful attributes. However adding new attributes to an existing file can incur the same expense as copying the file. See Chapter 9 [NetCDF File Structure and Performance], page 121, for a more extensive discussion.

2.5 Differences between Attributes and Variables

In contrast to variables, which are intended for bulk data, attributes are intended for ancillary data, or information about the data. The total amount of ancillary data associated with a netCDF object, and stored in its attributes, is typically small enough to be memory-resident. However variables are often too large to entirely fit in memory and must be split into sections for processing.

Another difference between attributes and variables is that variables may be multidimensional. Attributes are all either scalars (single-valued) or vectors (a single, fixed dimension).

Variables are created with a name, type, and shape before they are assigned data values, so a variable may exist with no values. The value of an attribute must be specified when it is created, so no attribute ever exists without a value.

A variable may have attributes, but an attribute cannot have attributes. Attributes assigned to variables may have the same units as the variable (for example, `valid_range`) or have no units (for example, `scale_factor`). If you want to store data that requires units different from those of the associated variable, it is better to use a variable than an attribute. More generally, if data require ancillary data to describe them, are multidimensional, require any of the defined netCDF dimensions to index their values, or require a significant amount of storage, that data should be represented using variables rather than attributes.

3 Data

This chapter discusses the six primitive netCDF data types, the kinds of data access supported by the netCDF interface, and how data structures other than arrays may be implemented in a netCDF file.

3.1 NetCDF Data Types

The current set of primitive types supported by the netCDF interface are:

`character`	8-bit characters intended for representing text.
`byte`	8-bit signed or unsigned integers (see discussion below).
`short`	16-bit signed integers.
`long`	32-bit signed integers.
`float`	32-bit IEEE floating-point.
`double`	64-bit IEEE floating-point.

Except for the added `byte` and the lack of unsigned types, netCDF supports the same primitive data types as C. The names for the primitive data types are reserved words in CDL, so the names of variables, dimensions, and attributes must not be type names.

It is currently possible to interpret `byte` data as either signed (-128 to 127) or unsigned (0 to 255). The current version of the netCDF library simply reads and writes 8-bit bytes without needing to know whether they are signed. However, the addition of packed data in a future version of netCDF will require arithmetic operations on values, and for that purpose `byte` data will be interpreted as *signed*.

These types were chosen because they are familiar to C and FORTRAN programmers, they have well-defined external representations independent of any particular computers (using XDR), and they are sufficient for providing a reasonably wide range of trade-offs between data precision and number of bits required for each datum. See Section 7.1 [Variables], page 60, for the correspondence between netCDF data types and the data types of a language.

There are plans for new data types, including 64-bit integers and n-bit packing.

3.2 Data Access

To access (read or write) netCDF data you specify an open netCDF file, a netCDF variable, and information (e.g. indices) identifying elements of the variable. In addition, the netCDF interface supports a form of record-oriented data access.

Access to data is *direct*, which means you can access a small subset of data from a large dataset efficiently, without first accessing all the data that precedes it. Reading and writing data by specifying a variable, instead of a position in a file, makes data access independent of how many other variables are in the file, making programs immune to data format changes that involve adding more variables to the data.

In the C and FORTRAN interfaces, files are not specified by name every time you want to access data, but instead by a small integer called a file ID, obtained when the file is first created or opened. Similarly, a variable is not specified by name for every data access either, but by a variable ID, a small integer used to identify a variable in a netCDF file. (In the C++ interface, open netCDF files and variables are objects, so no IDs are needed.)

3.2.1 Forms of Data Access

The netCDF interface supports several forms of direct access to data values in an open netCDF file. We describe each of these forms of access in order of increasing generality:

- access to individual elements, specified with an *index vector*;
- access to array sections, specified with an *index vector*, and *count vector*;
- access to subsampled array sections, specified with an *index vector*, *count vector*, and *stride vector*; and
- access to mapped array sections, specified with an *index vector*, *count vector*, *stride vector*, and an *index mapping vector*.

These four types of vector (*index vector*, *count vector*, *stride vector* and *index mapping vector*) are all vectors with an element for each dimension. For an n-dimensional variable (rank = n), an n-element vector is needed. If the variable is a scalar (no dimensions), these vectors are ignored.

An *array section* is a "slab" or contiguous rectangular block that is specified by two vectors. The *index vector* gives the indices of the element in the corner closest to the origin. The *count vector* gives the lengths of the edges of the slab along each of the variable's dimensions, in order. The number of values accessed is the product of these edge lengths.

A *subsampled array section* is similar to an *array section*, except that an additional *stride vector* is used specify sampling. This vector has an element for each dimension giving the length of the strides to be taken along that dimension. For example, a stride of 4 means every fourth value along the corresponding dimension. The total number of values accessed is the product of the ceiling (i.e. rounding up to integer) of each edge length (defined by the *count vector*) divided by the corresponding stride.

A *mapped array section* is similar to a *subsampled array section* except that an additional *index mapping vector* allows one to specify how data values associated with the netCDF variable are arranged in memory. The offset, in bytes, of each value from the reference location, is given by the sum of the products of each index by the corresponding element of the index mapping vector. The number of values accessed is the same as for a *subsampled array section*.

The use of mapped array sections is discussed more fully below, but first we present an example of the more commonly used array-section access.

3.2.2 An Example of Array-Section Access

Assume that in our earlier example netCDF file (see Chapter 2 [CDL example], page 13), we wish to read a cross-section of all the data for the `temp` variable at one level (say, the second), and assume that there are currently three records (`time` values) in the netCDF file. Recall that the dimensions are defined as

```
lat = 5, lon = 10, level = 4, time = unlimited;
```

and the variable `temp` is declared as

```
float   temp(time, level, lat, lon);
```

in the CDL notation.

A corresponding C variable that holds data for only one level might be declared as

```
#define LATS  5
#define LONS 10
#define LEVELS 1
#define TIMES 3                 /* currently */
   ...
float   temp[TIMES*LEVELS*LATS*LONS];
```

to keep the data in a one-dimensional array, or

```
   ...
float   temp[TIMES][LEVELS][LATS][LONS];
```

using a multidimensional array declaration.

In FORTRAN, the dimensions are reversed from the CDL declaration with the first dimension varying fastest and the record dimension as the last dimension of a record variable. Thus a FORTRAN declaration for the corresponding variable that holds data for only one level is

```
INTEGER LATS, LONS, LEVELS, TIMES
PARAMETER (LATS=5, LONS=10, LEVELS=1, TIMES=3)
  ...
REAL TEMP(LONS, LATS, LEVELS, TIMES)
```

To specify the block of data that represents just the second level, all times, all latitudes, and all longitudes, we need to provide a corner and some edge lengths. The corner should be (0, 1, 0, 0) in C—or (1, 1, 2, 1) in FORTRAN—because we want to start at the beginning of each of the `time`, `lon`, and `lat` dimensions, but we want to begin at the second value of the `level` dimension. The edge lengths should be (3, 1, 5, 10) in C—or (10, 5, 1, 3) in FORTRAN—since we want to get data for all three `time` values, only one `level` value, all five `lat` values, and all 10 `lon` values. We should expect to get a total of 150 float values returned (3 * 1 * 5 * 10), and should provide enough space in our array for this many. The order in which the data will be returned is with the last dimension, `lon`, varying fastest for C, or with the first dimension, `LON`, varying fastest for FORTRAN:

```
                C                       FORTRAN

        temp[0][1][0][0]        TEMP(1, 1, 2, 1)
        temp[0][1][0][1]        TEMP(2, 1, 2, 1)
        temp[0][1][0][2]        TEMP(3, 1, 2, 1)
        temp[0][1][0][3]        TEMP(4, 1, 2, 1)

              ...                     ...

        temp[2][1][4][7]        TEMP( 8, 5, 2, 3)
        temp[2][1][4][8]        TEMP( 9, 5, 2, 3)
        temp[2][1][4][9]        TEMP(10, 5, 2, 3)
```

Note that the different dimension orders for the C and FORTRAN interfaces do not reflect a different order for values stored on the disk, but merely different orders supported by the procedural interfaces to the two languages. In general, it does not matter whether a netCDF file is written using the C or FORTRAN interface; netCDF files written from either language may be read by programs written in the other language.

3.2.3 More on General Array Section Access

The use of mapped array sections allows non-trivial relationships between the disk addresses of variable elements and the addresses where they are stored in memory. For example, a matrix in memory could be the transpose of that on disk, giving a quite different order of elements. In a regular array section, the mapping between the disk and memory addresses is trivial: the structure of the in-memory values (i.e. the dimensional sizes and their order) is identical to that of the array section. In a mapped array section, however, an *index mapping vector* is used to define the mapping between indices of netCDF variable elements and their memory addresses. The offset, in bytes, from the origin of a memory-resident array to a particular point is given by the *inner product*[1] of the index mapping vector with the point's *index vector*[2] . The index mapping vector for a regular array section would have — in order from most rapidly varying dimension to most slowly — the byte size of a memory-resident datum (e.g. 4 for a floating-point value), then the product of that

[1] The *inner product* of two vectors [x0, x1, ..., xn] and [y0, y1, ..., yn] is just x0*y0 + x1*y1 + ... + xn*yn.

[2] A point's *coordinate offset vector* gives, for each dimension, the offset from the origin of the containing array to the point. In C, a point's coordinate offset vector is the same as it's coordinate vector. In FORTRAN, however, the values of a point's coordinate offset vector are one less than the corresponding values of the point's coordinate vector.

value with the edge length of the most rapidly varying dimension of the array section, then the product of that value with the edge length of the next most rapidly varying dimension, and so on. In a mapped array, however, the correspondence between netCDF variable disk locations and memory locations can be radically different. For example, the following C definitions

```
struct vel {
    int flags;
    float u;
    float v;
} vel[NX][NY];
long imap[2] = {
    sizeof(struct vel),
    sizeof(struct vel)*NY};
```

where `imap` is the index mapping vector, can be used to access the memory-resident values of the netCDF variable, `vel(NY,NX)`, even though the dimensions are transposed and the data is contained in a 2-D array of structures rather than a 2-D array of floating-point values.

A more detailed example of mapped array access is presented in the description of the C and FORTRAN interfaces for mapped array access. See Section 7.7 [Write a Subsampled Or Mapped Array of Values: ncvarputg, NCVPTG, and NCVPGC], page 73.

Note that, although the netCDF abstraction allows the use of subsampled or mapped array-section access if warranted by the situation, they are not required. If you do not need these more general forms of access, you may ignore these capabilities and use single value access or regular array section access instead.

3.2.4 Record-Oriented Access

Record-oriented access provides a more efficient alternative method in C (not FORTRAN) of reading or writing a whole record or part of a record. A record contains data for all the record variables, and any number of these can be read or written in a single record-oriented access call.

You specify a netCDF file, a record number (index of unlimited dimension) and an array of pointers to buffers (areas of memory) for each of the variables in the record. Those variables corresponding to NULL values in this array are ignored.

An example where the gain in speed could be considerable would be a file consisting of fifty variables, all of which have just one dimension which is the unlimited dimension. Thus each record contains a single value for each of fifty variables. It would be much faster to use a single record-oriented call, which reads or writes a whole record of fifty values, than to use fifty separate conventional calls, which each read or write a single value.

3.3 Data Structures

The only kind of data structure directly supported by the netCDF abstraction is a collection of named arrays with attached vector attributes. NetCDF is not particularly well-suited for storing linked lists, trees, sparse matrices, ragged arrays or other kinds of data structures requiring pointers. It is possible to build other kinds of data structures from sets of arrays by adopting various conventions regarding the use of data in one array as pointers into another array. The netCDF library won't provide much help or hindrance with constructing such data structures, but netCDF provides the mechanisms with which such conventions can be designed.

The following example stores a ragged array `ragged_mat` using an attribute `row_index` to name an associated index variable giving the index of the start of each row. The first row contains 12 (12-0) elements, the second 7 (19-12), etc.

```
        float   ragged_mat(max_elements);
                ragged_mat:row_index = "row_start";
        int     row_start(max_rows);
    data:
        row_start   = 0, 12, 19, ...
```

As another example, netCDF variables may be grouped within a netCDF file by defining attributes that list the names of the variables in each group, separated by a conventional delimiter such as a space or comma. A convention can be adopted to use particular sorts of attribute names for such groupings, so that an arbitrary number of named groups of variables can be supported. If needed, a particular conventional attribute for each variable might list the names of the groups of which it is a member. Use of attributes, or variables that refer to other attributes or variables, provides a flexible mechanism for representing some kinds of complex structures in netCDF files.

4 Use of the NetCDF Library

You can use the netCDF library without knowing about all of the netCDF interface. If you are creating a netCDF file, only a handful of routines are required to define the necessary dimensions, variables, and attributes, and to write the data to the netCDF file. (Even less are needed if you use the `ncgen` utility to create the file before running a program using netCDF library calls to write data.) Similarly, if you are writing software to access data stored in a particular netCDF object, only a small subset of the netCDF library is required to open the netCDF file and access the data. Authors of generic applications that access arbitrary netCDF files need to be familiar with more of the netCDF library.

In this chapter we provide templates of common sequences of netCDF subroutine calls needed for common uses. For clarity we present only the names of routines; omit declarations and error checking; indent statements that are typically invoked multiple times; and use ... to represent arbitrary sequences of other statements. Full argument lists for the procedures and subroutines are described in later chapters.

4.1 Creating a NetCDF File

The typical sequences of netCDF calls used to create a new netCDF file follows.

```
nccreate      /* create netCDF file: enter define mode */
     ...
   ncdimdef   /* define dimension: from name and size */
     ...
   ncvardef   /* define variable: from name, type, dimensions */
     ...
   ncattput   /* put attribute: assign attribute values */
     ...
ncendef       /* end definitions: leave define mode */
     ...
   ncvarput   /* put variable: provide values for variables */
     ...
ncclose       /* close: save new netCDF file */
```

In FORTRAN, the corresponding sequence looks like this:

```
NCCRE              ! create netCDF file: enter define mode
     ...
   NCDDEF          ! define dimensions: from name and size
     ...
   NCVDEF          ! define variables: from name, type, dimensions
     ...
   NCAPT or NCAPTC ! put attribute: assign attribute values
     ...
NCENDF             ! end definitions: leave define mode
     ...
```

```
           NCVPT or NCVPTC ! put variable: provide values for variables
             ...
        NCCLOS             ! close: save new netCDF file
```

Only one call is needed to create a netCDF file, at which point you will be in the first of two netCDF *modes*. When accessing an open netCDF file, it is either in *define mode* or *data mode*. In define mode, you can create dimensions, variables, and new attributes, but you cannot read or write variable data. In data mode, you can access data and change existing attributes, but you are not permitted to create new dimensions, variables, or attributes.

One call to `ncdimdef` (or `NCDDEF`) is needed for each dimension created. Similarly, one call to `ncvardef` (or `NCVDEF`) is needed for each variable creation, and one call to `ncattput` (or `NCAPT` or `NCAPTC`) is needed for each attribute defined and assigned a value. To leave define mode and enter data mode, call `ncendef` (or `NCENDF`).

Once in data mode, you can add new data to variables, change old values, and change values of existing attributes (so long as the attribute changes do not require more storage space). The FORTRAN interface provides two subroutines for defining attributes and providing values for variables, depending on whether a numeric or character string value is used. Single values are written to a variable with `ncvarput1` (or `NCVPT1` or `NCVP1C`); while arbitrary arrays of data are written using `ncvarput` or `ncvarputg` (or `NCVPT`, `NCVPTC`, `NCVPTG`, or `NCVPGC`) instead. Multi-variable records of data may be written using multiple calls to `ncvarput` (or `NCVPT`) or with a single call to `ncrecput`.

Finally, you should explicitly close all netCDF files that have been opened for writing by calling `ncclose` (or `NCCLOS`). If a program terminates abnormally with netCDF files open for writing, you may lose one or more records of the most recently written record variable data as well as any attribute changes since the last call to `ncsync` (or `NCSNC`). It is possible to reduce the chance of losing data due to abnormal termination by explicitly calling `ncsync` (`NCSNC`) after every write to netCDF variables or change to attribute values. This can be expensive in computer resources, so such calls should ordinarily be omitted unless they are really needed.

4.2 Reading a NetCDF File with Known Names

Here we consider the case where you know the names of not only the netCDF files, but also the names of their dimensions, variables, and attributes. (Otherwise you would have to do "inquire" calls.) The order of typical C calls to read data from those variables in a netCDF file is:

```
        ncopen           /* open existing netCDF file */
          ...
           ncdimid       /* get dimension IDs */
             ...
           ncvarid       /* get variable IDs */
             ...
           ncattget      /* get attribute values */
```

```
         ...
       ncvarget       /* get values of variables */
         ...
    ncclose           /* close netCDF file */
```

In FORTRAN, the corresponding sequence looks like this:

```
    NCOPN                 !  open existing netCDF file
         ...
       NCDID              !  get dimension IDs
         ...
       NCVID              !  get variable IDs
         ...
       NCAGT or NCAGTC    !  get attribute values
         ...
       NCVGT or NCVGTC    !  get values of variables
         ...
    NCCLOS                !  close netCDF file
```

First, a single call opens the netCDF file, given the file name, and returns a netCDF ID that is used to refer to the open netCDF file in all subsequent calls.

Next, a call to `ncdimid` (or `NCDID`) for each dimension of interest gets the dimension ID from the dimension name. Similarly, each required variable ID is determined from its name by a call to `ncvarid` (or `NCVID`). Once variable IDs are known, variable attribute values can be retrieved using the netCDF ID, the variable ID, and the desired attribute name as input to `ncattget` (or `NCAGT` or `NCAGTC`) for each desired attribute. Variable data values can be directly accessed from the netCDF file with `ncvarget1` (or `NCVGT1` or `NCVG1C`) for single values, `ncvarget` or `ncvargetg` (or `NCVGT`, `NCVGTC`, `NCVGTG`, or `NCVGGC`) for cross-sections of values, or `ncrecget` for records of values.

Finally, the netCDF file is closed with `ncclose` (or `NCCLOS`). There is no need to close a file open only for reading.

4.3 Reading a netCDF File with Unknown Names

Many programs (e.g. generic software) need to get It is possible to write programs (e.g. generic software) which do such things as processing every variable (except perhaps coordinate variables) in some way, without needing to know in advance the names of these variables. Even the names of dimensions and attributes may be unknown.

Names and other information may be obtained from netCDF files by calling inquire functions. These return information about a whole netCDF file, a dimension, a variable, or an attribute. The following template illustrates how they are used:

```
ncopen          /* open existing netCDF file */
  ...
ncinquire       /* find out what is in it */
     ...
   ncdiminq     /* get dimension names, sizes */
     ...
   ncvarinq     /* get variable names, types, shapes */
        ...
      ncattname /* get attribute names */
        ...
      ncattinq  /* get attribute types and lengths */
        ...
      ncattget  /* get attribute values */
        ...
   ncvarget     /* get values of variables */
     ...
ncclose         /* close netCDF file */
```

In FORTRAN, the corresponding sequence looks like this:

```
NCOPN                ! open existing netCDF file
  ...
NCINQ                ! find out what is in it
     ...
   NCDINQ            ! get dimension names, sizes
     ...
   NCVINQ            ! get variable names, types, shapes
        ...
      NCANAM         ! get attribute names
        ...
      NCAINQ         ! get attribute values
        ...
      NCAGT or NCAGTC ! get attribute values
        ...
   NCVGT or NCVGTC   ! get values of variables
     ...
NCCLOS               ! close netCDF file
```

As in the previous example, a single call opens the existing netCDF file, returning a netCDF ID. This netCDF ID is given to the `ncinquire` (or `NCINQ`) routine, which returns the number of dimensions, the number of variables, the number of global attributes, and the ID of the unlimited dimension, if there is one.

Another inquire function, `ncrecinq`, provides a convenient way of obtaining information about record variables, although the information can also be obtained using the other inquire functions. The `ncrecinq` function returns the number of record variables, their variable IDs, and how much memory is needed for a data record.

All the inquire functions are quite inexpensive to use and require no I/O, since the information they provide is stored in memory. In the C interface, the inquire functions also support getting a subset of information, by providing null pointers instead of valid addresses, for undesired information.

Dimension IDs are assigned by using consecutive integers (beginning at 0 in C, 1 in FORTRAN). Also dimensions, once created, cannot be deleted. Therefore, knowing the number of dimension IDs in a netCDF file means knowing all the dimension IDs: they are the integers 0, 1, 2, ..., (or 1, 2, 3, ... in FORTRAN). For each dimension ID, a call to the inquire function `ncdiminq` (or `NCDINQ`) returns the dimension name and size.

Variable IDs are also assigned from consecutive integers 0, 1, 2, ..., (or 1, 2, 3, ... in FORTRAN). These can be used in `ncvarinq` (or `NCVINQ`) calls to find out the names, types, shapes, and the number of attributes assigned to each variable.

Once the number of attributes for a variable is known, successive calls to `ncattname` (or `NCANAM`) return the name for each attribute given the netCDF ID, variable ID, and attribute number. Armed with the attribute name, a call to `ncattinq` (or `NCAINQ`) returns its type and length. Given the type and length, you can allocate enough space to hold the attribute values. Then a call to `ncattget` (or `NCAGT` or `NCAGTC`) returns the attribute values.

Once the names, IDs, types, shapes, and lengths of all netCDF components are known, data values can be accessed by calling `ncvarget1` (or `NCVGT1` or `NCVG1C`) for single values, `ncvarget` or `ncvargetg` (or `NCVGT`, `NCVGTC`, `NCVGTG`, or `NCVGGC`) for aggregates of values using array access, or `ncrecget` for aggregates of values using record access.

4.4 Adding New Dimensions, Variables, Attributes

An existing netCDF file can be extensively altered. New dimensions, variables, and attributes can be added or existing ones renamed, and existing attributes can be deleted. Existing dimensions, variables, and attributes can be renamed. The following code template lists a typical sequence of calls to add new netCDF components to an existing file:

```
ncopen           /* open existing netCDF file */
  ...
ncredef          /* put it into define mode */
    ...
  ncdimdef       /* define additional dimensions (if any) */
    ...
  ncvardef       /* define additional variables (if any) */
    ...
  ncattput       /* define additional attributes (if any) */
    ...
ncendef          /* check definitions, leave define mode */
    ...
  ncvarput       /* provide values for new variables */
```

```
      ...
    ncclose         /* close netCDF file */
```

In FORTRAN, the corresponding sequence looks like this:

```
    NCOPN             !  open existing netCDF file
      ...
    NCREDF            !  put it into define mode
      ...
      NCDDEF          !  define additional dimensions (if any)
      ...
      NCVDEF          !  define additional variables (if any)
      ...
      NCAPT or NCAPTC !  define additional attributes (if any)
      ...
    NCENDF            !  check definitions, leave define mode
      ...
      NCVPT or NCVPTC !  provide values for new variables
      ...
    NCCLOS            !  close netCDF file
```

A netCDF file is first opened by the `ncopen` (or `NCOPN`) call. This call puts you in *data mode*, which means existing data values can be accessed and changed, existing attributes can be changed (so long as they do not grow), but nothing can be added. To add new netCDF dimensions, variables, or attributes you must leave data mode and enter *define mode*, by calling `ncredef` (or `NCREDF`). In define mode, call `ncdimdef` (or `NCDDEF`) to define new dimensions, `ncvardef` (or `NCVDEF`) to define new variables, and `ncattput` (or `NCAPT` or `NCAPTC`) to assign new attributes to variables or enlarge old attributes.

You can leave define mode and reenter data mode, checking all the new definitions for consistency and committing the changes to disk, by calling `ncendef` (or `NCENDF`). If you do not wish to reenter data mode, just call `ncclose` (or `NCCLOS`), which will have the effect of first calling `ncendef` (or `NCENDF`).

Until the `ncendef` (or `NCENDF`) call, you may back out of all the redefinitions made in define mode and restore the previous state of the netCDF dataset by calling `ncabort` (or `NCABOR`). You may also use the `ncabort` call to restore the netCDF dataset to a consistent state if the call to `ncendef` (or `NCENDF`) fails. If you have called `ncclose` (or `NCCLOS`) from definition mode and the implied call to `ncendef` (or `NCENDF`) fails, `ncabort` (or `NCABOR`) will automatically be called to close the netCDF file and leave it in its previous consistent state (before you entered define mode).

4.5 Error Handling

The netCDF library provides the facilities needed to handle errors in a flexible way. However it is often unnecessary to specify anything about error handling, since by default all netCDF library routines just print an error message and exit when an error occurs. If this (admittedly drastic) error behavior is acceptable, you never need to check return values, since any condition that would result in an error will print an explanatory message and exit. For simplicity, the examples in this guide assume this default error-handling behavior, so there is no checking of return values.

In the C interface, errors may be handled more flexibly by setting the external integer `ncopts`, declared in the file '`netcdf.h`'. Two aspects of the error-handling behavior can be modified independently: the suppression of error messages, and the fatality of errors. The default behavior, that errors are both verbose and fatal, is specified by the assignment

```
ncopts = NC_VERBOSE | NC_FATAL;
```

where `NC_VERBOSE` and `NC_FATAL` are predefined constants from the include file '`netcdf.h`'.

If you want error messages but do not wish errors to be fatal, turn off the fatal error flag with:

```
ncopts = NC_VERBOSE;
```

If you want neither error messages nor fatal errors, turn off both flags with:

```
ncopts = 0;
```

In non-fatal mode you should check the return value after each call to a netCDF function. This value is `0` normally and `-1` if an error occurs. Another externally-defined integer, `ncerr`, contains a netCDF-specific error code that is available after an error has occurred to determine the nature of the error. The names and descriptions of netCDF error codes are included in the file '`netcdf.h`'.

In the FORTRAN interface, the error options described above can be accessed by using the routines NCPOPT and NCGOPT. The default error- handling behavior is equivalent to the statement

```
      CALL NCPOPT(NCVERBOS+NCFATAL)
```

where the values of NCVERBOS and NCFATAL are predefined constants from the FORTRAN include file '`netcdf.inc`'. If you want error messages, but do not wish errors to be fatal, turn off the fatal error flag with:

```
      CALL NCPOPT(NCVERBOS)
```

If you want neither error messages nor fatal errors, turn off both flags with:

```
      CALL NCPOPT(0)
```

To get the current value of the error options, use:

```
      CALL NCGOPT(NCOPTS)
```

In either case, the integer return code (the last parameter in all of the FORTRAN subroutines and functions) contains the non-zero netCDF-specific error number that can be used to determine the nature of the error. Names and descriptions of netCDF error codes are included in the file '`netcdf.inc`'.

Occasionally, low-level write errors may occur in the XDR library layer below the netCDF library. For example, if a write operation causes you to exceed disk quotas or to attempt to write to a device that is no longer available, you may get an error message from one of the XDR functions rather than from the netCDF library.

4.6 Compiling and Linking with the NetCDF Library

Details of how to compile and link a program that uses the netCDF C or FORTRAN interfaces differ, depending on the operating system, the available compilers, and where the netCDF library and include files are installed. Nevertheless, we provide here examples of how to compile and link a program that uses the netCDF library on a Unix platform, so that you can adjust these examples to fit your installation.

C Interface

Every C file that references netCDF functions or constants must contain an appropriate `#include` statement before the first such reference:

```
#include <netcdf.h>
```

Unless the '`netcdf.h`' file is installed in a standard directory where the C compiler always looks, you must use the `-I` option when invoking the compiler, to specify a directory where '`netcdf.h`' is installed, for example:

```
cc -c -I/usr/local/netcdf/include myprogram.c
```

Alternatively, you could specify an absolute pathname in the `#include` statement, but then your program would not compile on another platform where netCDF is installed in a different location.

Unless the netCDF library is installed in a standard directory where the linker always looks, you must use the `-L` and `-l` options to link an object file that uses the netCDF library. For example:

```
cc -o myprogram myprogram.o -L/usr/local/netcdf/lib -lnetcdf
```

Alternatively, you could specify an absolute pathname for the library:

```
cc -o myprogram myprogram.o -l/usr/local/netcdf/lib/libnetcdf.a
```

On some systems, you must specify an additional library after the netCDF library where the system XDR libraries are found.

FORTRAN Interface

Every FORTRAN file that references netCDF functions or constants must contain an appropriate `INCLUDE` statement before the first such reference:

```
INCLUDE 'netcdf.inc'
```

Unless the '`netcdf.inc`' file is installed in a standard directory where the FORTRAN compiler always looks, you must use the `-I` option when invoking the compiler, to specify a directory where '`netcdf.inc`' is installed, for example:

```
f77 -c -I/usr/local/netcdf/include myprogram.f
```

Alternatively, you could specify an absolute pathname in the `INCLUDE` statement, but then your program would not compile on another platform where netCDF is installed in a different location.

Unless the netCDF library is installed in a standard directory where the linker always looks, you must use the `-L` and `-l` options to link an object file that uses the netCDF library. For example:

```
f77 -o myprogram myprogram.o -L/usr/local/netcdf/lib -lnetcdf
```

Alternatively, you could specify an absolute pathname for the library:

```
f77 -o myprogram myprogram.o -l/usr/local/netcdf/lib/libnetcdf.a
```

On some systems, you must specify an additional library after the netCDF library where the system XDR libraries are found.

5 Files

This chapter presents the interfaces of the netCDF routines that deal with a netCDF file as a whole.

A netCDF file that has not yet been opened can only be referred to by its file name. Once a netCDF file is opened, it is referred to by a *netCDF ID*, which is a small nonnegative integer returned when you create or open the file. A netCDF ID is much like a file descriptor in C or a logical unit number in FORTRAN. In any single program, the netCDF IDs of distinct open netCDF files are distinct. A single netCDF file may be opened multiple times and will then have multiple distinct netCDF IDs; however at most one of the open instances of a single netCDF file should permit writing. When an open netCDF file is closed, the ID is no longer associated with a netCDF file.

The operations supported on a netCDF file as a single object are:

- Create, given file name and whether to overwrite or not.
- Open for access, given file name and read or write intent.
- Put into define mode, to add dimensions, variables, or attributes.
- Take out of define mode, checking consistency of additions.
- Close, writing to disk if required.
- Get number of dimensions, number of variables, number of global attributes, and ID of the unlimited dimension, if any.
- Synchronize to disk to make sure it is current.
- Set and unset *nofill* mode for optimized sequential writes.

After a summary of conventions used in describing the netCDF C and FORTRAN interfaces, the rest of this chapter presents the interfaces for these operations.

5.1 NetCDF Library Interface Descriptions

Each interface description for a particular netCDF function in this and later chapters contains:

- A description of the purpose of the function;
- A list of possible error conditions;
- A C function prototype that presents the type and order of the formal parameters to the function;
- A description of each formal parameter in the C interface;
- An example of a C program fragment calling the netCDF function and perhaps other netCDF functions;
- A FORTRAN function prototype that presents the type and order of the formal parameters to the FORTRAN function or functions that provide the same functionality as the C function;

- A description of each formal parameter in the FORTRAN interface; and
- An example of a FORTRAN program fragment that duplicates the function of the example C fragment.

The C function prototypes specify the order and type of each formal parameter and conform to the ANSI C standard. For FORTRAN a similar syntax is used to concisely present the order and types of FORTRAN formal parameters. In the few cases in which a single C function corresponds to two FORTRAN functions, the FORTRAN functions prototypes are presented together.

5.2 Create a NetCDF file: nccreate and NCCRE

The function `nccreate` (or `NCCRE` for FORTRAN) creates a new netCDF file, returning a netCDF ID that can subsequently be used to refer to the netCDF file. The new netCDF file is placed in define mode.

In case of an error, `nccreate` returns -1; `NCCRE` returns a nonzero value in `rcode`. Possible causes of errors include::

- Passing a file name that includes a directory that does not exist.
- Specifying a file name of a file that exists and also specifying `NC_NOCLOBBER` (or `NCNOCLOB`).
- Attempting to create a netCDF file in a directory where you don't have permission to create files.

C Interface: nccreate

```
int nccreate (const char* filename, int cmode);
```

`filename` The file name of the new netCDF file.

`cmode` Should be specified as either `NC_CLOBBER` or `NC_NOCLOBBER`. These constants are defined in the include file named '`netcdf.h`'. `NC_CLOBBER` means that even if the file already exists, you want to create a new file with the same name, erasing the old file's contents. `NC_NOCLOBBER` means you want to create a new netCDF file only if the given file name does not refer to a file that already exists.

In this example we create a netCDF file named '`foo.nc`'; we want the file to be created in the current directory only if a file with that name does not already exist:

```
#include <netcdf.h>
   ...
int ncid;
   ...
ncid = nccreate("foo.nc", NC_NOCLOBBER);
```

FORTRAN Interface: NCCRE

```
INTEGER FUNCTION NCCRE (CHARACTER*(*) FILENAME, INTEGER CMODE,
                        INTEGER RCODE)
```

`FILENAME` The file name of the new netCDF file.

`CMODE` Should be specified as either `NCCLOB` or `NCNOCLOB`. These constants are defined in the include file '`netcdf.inc`'. `NCCLOB` means that even if the file already exists, you want to create a new file with the same name, erasing the old file's contents. `NCNOCLOB` means you want to create a new netCDF file only if the given file name does not refer to a file that already exists.

`RCODE` Returned error code. If no errors occurred, 0 is returned.

In this example we create a netCDF file named '`foo.nc`', assuming we want the file to be created in the current directory only if a file with that name does not already exist:

```
INCLUDE 'netcdf.inc'
   ...
INTEGER NCID
   ...
NCID = NCCRE('foo.nc', NCNOCLOB, RCODE)
```

5.3 Open a NetCDF File for Access: ncopen and NCOPN

The function `ncopen` (or `NCOPN` for FORTRAN) opens an existing netCDF file for access.

In case of an error, `ncopen` returns -1; `NCOPN` returns a nonzero value in `rcode`. Possible causes of errors include::

- The specified netCDF file does not exist.
- The mode specified is something other than `NC_WRITE` or `NC_NOWRITE`.

C Interface: ncopen

```
int ncopen(const char* filename,int mode);
```

`filename` File name for netCDF file to be opened.

`mode` Either `NC_WRITE`, to open the file for writing, or `NC_NOWRITE`, to open the file read-only. "Writing" means any kind of change to the file, including appending or changing data, adding or renaming dimensions, variables, and attributes, or deleting attributes.

Here is an example using `ncopen` to open an existing netCDF file named '`foo.nc`' for reading:

```
#include <netcdf.h>
   ...
int ncid;
   ...
ncid = ncopen("foo.nc", NC_NOWRITE);
```

FORTRAN Interface: NCOPN

```
      INTEGER FUNCTION NCOPN(CHARACTER*(*) FILENAME,
     +                       INTEGER RWMODE,
     +                       INTEGER RCODE)
```

`FILENAME` File name for netCDF file to be opened.

`RWMODE` Either `NCWRITE`, to open the file for writing, or `NCNOWRIT`, to open the file read-only. "Writing" means any kind of change to the file, including appending or changing data, adding or renaming dimensions, variables, and attributes, or deleting attributes.

`RCODE` Returned error code. If no errors occurred, 0 is returned.

Here is an example of using `NCOPN` to open an existing netCDF file named '`foo.nc`' for reading:

```
      INCLUDE 'netcdf.inc'
         ...
      INTEGER NCID
         ...
      NCID = NCOPN('foo.nc', NCNOWRIT, RCODE)
```

5.4 Put Open NetCDF File into Define Mode: ncredef and NCREDF

The function `ncredef` (or `NCREDF` for FORTRAN) puts an open netCDF file into define mode, so dimensions, variables, and attributes can be added or renamed and attributes can be deleted.

In case of an error, `ncredef` returns -1; `NCREDF` returns a nonzero value in `rcode`. Possible causes of errors include::

- The specified netCDF file is already in define mode.
- The specified netCDF file was opened for read-only.
- The specified netCDF ID does not refer to an open netCDF file.

C Interface: ncredef

```
int ncredef(int ncid);
```

`ncid` netCDF ID, returned from a previous call to `ncopen` or `nccreate`.

Here is an example using `ncredef` to open an existing NetCDF file named '`foo.nc`' and put it into define mode:

```
#include <netcdf.h>
   ...
int ncid;
   ...
ncid = ncopen("foo.nc", NC_WRITE);     /* open file */
   ...
ncredef(ncid);                         /* put in define mode */
```

FORTRAN Interface: NCREDF

```
SUBROUTINE NCREDF(INTEGER NCID, INTEGER RCODE)
```

`NCID`	NetCDF ID, returned from a previous call to `NCOPN` or `NCCRE`.
`RCODE`	Returned error code. If no errors occurred, 0 is returned.

Here is an example of using `NCREDF` to open an existing netCDF file named '`foo.nc`' and put it into define mode:

```
      INCLUDE 'netcdf.inc'
         ...
      INTEGER NCID
         ...
      NCID = NCOPN('foo.nc', NCWRITE, RCODE)
         ...
      CALL NCREDF(NCID, RCODE)
```

5.5 Leave Define Mode: ncendef and NCENDF

The function `ncendef` (or `NCENDF` for FORTRAN) takes an open netCDF file out of define mode. The changes made to the netCDF file while it was in define mode are checked and committed to disk if no problems occurred. The netCDF file is then placed in data mode, so variable data can be read or written.

This call can be expensive, since it involves initializing non-record variables and copying data under some circumstances. See Chapter 9 [NetCDF File Structure and Performance], page 121, for a more extensive discussion.

In case of an error, `ncendef` returns -1; `NCENDF` returns a nonzero value in `rcode`. Possible causes of errors include:

- The specified netCDF file is not in define mode.
- The specified netCDF ID does not refer to an open netCDF file.

C Interface: ncendef

```
int ncendef(int ncid);
```

`ncid` NetCDF ID, returned from a previous call to `ncopen` or `nccreate`.

Here is an example using `ncendef` to finish the definitions of a new netCDF file named '`foo.nc`' and put it into data mode:

```
#include <netcdf.h>
   ...
int ncid;
   ...
ncid = nccreate("foo.nc", NC_NOCLOBBER);

   ...      /* create dimensions, variables, attributes */

ncendef(ncid);  /*leave define mode*/
```

FORTRAN Interface: NCENDF

```
SUBROUTINE NCENDF(INTEGER NCID, INTEGER RCODE)
```

`NCID` NetCDF ID, returned from a previous call to `NCOPN` or `NCCRE`.

`RCODE` Returned error code. If no errors occurred, 0 is returned.

Here is an example using `NCENDF` to finish the definitions of a new netCDF file named '`foo.nc`' and put it into data mode:

```
      INCLUDE 'netcdf.inc'
         ...
      INTEGER NCID
         ...
      NCID = NCCRE('foo.nc', NCNOCLOB, RCODE)

         ...  !  create dimensions, variables, attributes

      CALL NCENDF(NCID, RCODE)
```

5.6 Close an Open NetCDF File: ncclose and NCCLOS

The function `ncclose` (or `NCCLOS` for FORTRAN) closes an open netCDF file. If the file is in define mode, `ncendef` (or `NCENDF`) will be called before closing. (In this case, if `ncendef` [or `NCENDF`] returns an error, `ncabort` [or `NCABOR`] will automatically be called to restore the file to the consistent state before define mode was last entered.) After an open netCDF file is closed, its netCDF ID will be reassigned to the next netCDF file that is opened or created.

In case of an error, `ncclose` returns -1; `NCCLOS` returns a nonzero value in `rcode`. Possible causes of errors include:

- Define mode was entered and the automatic call made to `ncendef` (or `NCENDF`) failed.
- The specified netCDF ID does not refer to an open netCDF file.

C Interface: ncclose

```
int ncclose(int ncid);
```

`ncid`	NetCDF ID, returned from a previous call to `ncopen` or `nccreate`.

Here is an example using `ncclose` to finish the definitions of a new netCDF file named '`foo.nc`' and release its netCDF ID:

```
#include <netcdf.h>
   ...
int ncid;
   ...
ncid = nccreate("foo.nc", NC_NOCLOBBER);

   ...      /* create dimensions, variables, attributes */

ncclose(ncid);       /* close netCDF file */
```

FORTRAN Interface: NCCLOS

```
SUBROUTINE NCCLOS(INTEGER NCID, INTEGER RCODE)
```

`NCID`	netCDF ID, returned from a previous call to `NCOPN` or `NCCRE`.
`RCODE`	Returned error code. If no errors occurred, 0 is returned.

Here is an example using `NCCLOS` to finish the definitions of a new netCDF file named '`foo.nc`' and release its netCDF ID:

```
      INCLUDE 'netcdf.inc'
         ...
      INTEGER NCID, RCODE
         ...
      NCID = NCCRE('foo.nc', NCNOCLOB, RCODE)

         ...  ! create dimensions, variables, attributes

      CALL NCCLOS(NCID, RCODE)
```

5.7 Inquire about an Open NetCDF File: ncinquire and NCINQ

The function `ncinquire` (`NCINQ` for FORTRAN) returns information about an open netCDF file, given its netCDF ID. It can be called from either define mode or data mode. It returns values for the number of dimensions, the number of variables, the number of global attributes, and the dimension ID of the dimension defined with unlimited size, if any. No I/O is required when this or any other 'inquire' function in the netCDF interface is called, since the functions merely return information that is stored in a table for each open netCDF file.

In case of an error, `ncinquire` returns -1; `NCINQ` returns a nonzero value in `rcode`. Possible cause of errors includes:

- The specified netCDF ID does not refer to an open netCDF file.

C Interface: ncinquire

```
int ncinquire(int ncid, int* ndims, int* nvars, int* ngatts,
                  int* recdim);
```

`ncid` NetCDF ID, returned from a previous call to `ncopen` or `nccreate`.

`ndims` Returned number of dimensions defined for this netCDF file. If this parameter is given as '`0`' (a null pointer), the number of dimensions will not be returned so no variable to hold this information needs to be declared.

`nvars` Returned number of variables defined for this netCDF file. If this parameter is given as '`0`' (a null pointer), the number of variables will not be returned so no variable to hold this information needs to be declared.

`ngatts` Returned number of global attributes defined for this netCDF file. If this parameter is given as '`0`' (a null pointer), the number of global attributes will not be returned so no variable to hold this information needs to be declared.

`recdim` Returned ID of the unlimited dimension, if there is one for this netCDF file. If no unlimited size dimension has been defined, -1 is returned for the value of `recdim`. If this parameter is given as '`0`' (a null pointer), the record dimension ID will not be returned so no variable to hold this information needs to be declared.

Here is an example using `ncinquire` to find out about a netCDF file named '`foo.nc`':

```
#include <netcdf.h>
   ...
int ncid, ndims, nvars, ngatts, recdim;
   ...
ncid = ncopen("foo.nc", NC_NOWRITE);
   ...
ncinquire(ncid, &ndims, &nvars, &ngatts, &recdim);
```

FORTRAN Interface: NCINQ

```
      SUBROUTINE NCINQ(INTEGER NCID, INTEGER NDIMS, INTEGER NVARS,
     *                 INTEGER NGATTS, INTEGER RECDIM, INTEGER RCODE)
```

`NCID`	NetCDF ID, returned from a previous call to `NCOPN` or `NCCRE`.
`NDIMS`	Returned number of dimensions defined for this netCDF file.
`NVARS`	Returned number of variables defined for this netCDF file.
`NGATTS`	Returned number of global attributes defined for this netCDF file.
`RECDIM`	Returned ID of the unlimited dimension, if there is one for this netCDF file. If no unlimited size dimension has been defined, -1 is returned for the value of `RECDIM`.
`RCODE`	Returned error code. If no errors occurred, 0 is returned.

Here is an example using `NCINQ` to find out about a netCDF file named '`foo.nc`':

```
      INCLUDE 'netcdf.inc'
         ...
      INTEGER NCID, NDIMS, NVARS, NATTS, RECDIM, RCODE
         ...
      NCID = NCOPN('foo.nc', NCNOWRIT, RCODE)
         ...
      CALL NCINQ(NCID, NDIMS, NVARS, NATTS, RECDIM, RCODE)
```

5.8 Synchronize an Open NetCDF File to Disk: ncsync and NCSNC

The function `ncsync` (or `NCSNC` for FORTRAN) makes sure that the disk copy of a netCDF file open for writing is current. The netCDF file must be in data mode. A netCDF file in define mode is synchronized to disk only when `ncendef` (or `NCENDF`) is called. A process that is reading a netCDF file that another process is writing can also call `ncsync` (or `NCSNC` for FORTRAN) to get updated with the changes made by the writing process (e.g. the number of records written), without having to close and reopen the file.

It can be expensive in computer resources to always synchronize to disk after every write of variable data or change of an attribute value. There are two reasons you might want to synchronize after writes:

- To minimize data loss in case of abnormal termination, or
- To make data available to other processes for reading immediately after it is written. But note that a process that already had the file open for reading would not see the number of records increase when the writing process calls `ncsync`; to accomplish this, the reading process must call `ncsync`.

Data is automatically synchronized to disk when a netCDF file is closed, or whenever you leave define mode.

In case of an error, `ncsync` returns -1; `NCSNC` returns a nonzero value in `rcode`. Possible causes of errors include:

- The netCDF file is in define mode.
- The specified netCDF ID does not refer to an open netCDF file.

C Interface: ncsync

```
int ncsync(int ncid);
```

`ncid`	NetCDF ID, returned from a previous call to `ncopen` or `nccreate`.

Here is an example using `ncsync` to synchronize the disk writes of a netCDF file named '`foo.nc`':

```
#include <netcdf.h>
   ...
int ncid;
   ...
ncid = ncopen("foo.nc", NC_WRITE);  /* open for writing */

   ...          /* write data or change attributes */

ncsync(ncid);       /* synchronize to disk */
```

FORTRAN Interface: NCSNC

```
      SUBROUTINE NCSNC(INTEGER NCID, INTEGER RCODE)
```

`NCID`	NetCDF ID, returned from a previous call to `NCOPN` or `NCCRE`.
`RCODE`	Returned error code. If no errors occurred, 0 is returned.

Here is an example using `NCSNC` to synchronize the disk writes of a netCDF file named '`foo.nc`':

```
      INCLUDE 'netcdf.inc'
         ...
      INTEGER NCID, RCODE
         ...
      NCID = NCOPN('foo.nc', NCWRITE, RCODE)
         ...
* write data or change attributes
         ...
      CALL NCSNC(NCID, RCODE)
```

5.9 Back Out of Recent Definitions: ncabort and NCABOR

The function `ncabort` (or `NCABOR` for FORTRAN), if not in define mode, closes the netCDF file. If the file is being created and is still in define mode, the file is deleted. If define mode was entered by a call to `ncredef` (or `NCREDF`), the netCDF file is restored to its state before definition mode was entered and the file is closed. The main reason for calling `ncabort` (or `NCABOR`) is to restore the netCDF file to a known consistent state in case anything goes wrong during the definition of new dimensions, variables, or attributes.

This function is called automatically if `ncclose` (or `NCCLOS`) is called from define mode and the call to leave define mode before closing fails.

In case of an error, `ncabort` returns -1; `NCABOR` returns a nonzero value in `rcode`. Possible causes of errors include:

- When called from define mode while creating a netCDF file, deletion of the file failed.
- The specified netCDF ID does not refer to an open netCDF file.

C Interface: ncabort

```
int ncabort(int ncid);
```

`ncid` NetCDF ID, returned from a previous call to `ncopen` or `nccreate`.

Here is an example using `ncabort` to back out of redefinitions of a file named '`foo.nc`':

```
#include <netcdf.h>
   ...
int ncid;
   ...
ncid = ncopen("foo.nc", NC_WRITE);  /* open for writing */
   ...
ncredef(ncid);                     /* enter define mode */
   ...
if (ncdimdef(ncid, "lat", 18L) == -1)
   ncabort(ncid);                  /* define failed, abort */
```

FORTRAN Interface: NCABOR

```
      SUBROUTINE NCABOR(INTEGER NCID, INTEGER RCODE)
```

`NCID` NetCDF ID, returned from a previous call to `NCOPN` or `NCCRE`.

`RCODE` Returned error code. If no errors occurred, 0 is returned.

Here is an example using `NCABOR` to back out of redefinitions of a file named '`foo.nc`':

```
INCLUDE 'netcdf.inc'
   ...
INTEGER NCID, RCODE, LATID
   ...
NCID = NCOPN('foo.nc', NCWRITE, RCODE)
   ...
CALL NCREDF(NCID, RCODE)
   ...
LATID = NCDDEF(NCID, 'LAT', 18, RCODE)
IF (RCODE .EQ. -1) THEN  ! dimension definition failed
   CALL NCABOR(NCID, RCODE)  ! abort redefinitions
ENDIF
   ...
```

5.10 Set Fill Mode for Writes: ncsetfill and NCSFIL

These calls are intended for advanced usage, to optimize writes under some circumstances described below. The function `ncsetfill` (or `NCSFIL` for FORTRAN) sets the *fill mode* for an netCDF file open for writing and returns the current fill mode. The fill mode can be specified as either `NC_FILL` or `NC_NOFILL` (`NCFILL` or `NCNOFILL` for FORTRAN). The default behavior corresponding to `NC_FILL` is that data is pre-filled with fill values, that is fill values are written when you create non-record variables or when you write a value beyond data that hasn't been written yet. This makes it possible to detect attempts to read data before it was written. See Section 7.14 [Fill Values], page 94, for more information on the use of fill values. See Section 8.1 [Attribute Conventions], page 101, for information about how to define your own fill values.

The behavior corresponding to `NC_NOFILL` overrides the default behavior of prefilling data with fill values. This can be used to enhance performance, because it avoids the duplicate writes that occur when the netCDF library writes fill values that are immediately overwritten with data.

A value indicating which mode the netCDF file was already in is returned. You can use this value to temporarily change the fill mode of an open netCDF file and then restore it to the previous mode.

After you turn on `NC_NOFILL` mode for an open netCDF file, you must be certain to write valid data in all the positions that will later be read. Note that `NC_NOFILL` mode is only a transient property of a netCDF file open for writing: if you close and reopen the file, it will revert to the default behavior. You can also revert to the default behavior by calling `ncsetfill` (or `NCSFIL` for FORTRAN) again to explicitly set the fill mode to `NC_FILL`.

There are three situations where it is advantageous to set nofill mode:

1. Creating and initializing a netCDF file. In this case, you should set nofill mode before calling `ncendef` (`NCENDF` for FORTRAN), and then write *completely* all non-record variables and the initial records of all the record variables you want to initialize.

2. Extending an existing record-oriented netCDF file. Set nofill mode after opening the file for writing, then append the additional records to the file completely, leaving no intervening unwritten records.
3. Adding new variables that you are going to initialize to an existing netCDF file. Set nofill mode before calling `ncendef` (`NCENDF` for FORTRAN), then write all the new variables completely.

If the netCDF file has an unlimited dimension and the last record was written while in `NC_NOFILL` mode, then the file will be 4 bytes longer than if `NC_NOFILL` mode wasn't set, but this will be completely transparent if you access the data only through the netCDF interfaces.

In case of an error, `ncsetfill` returns -1; `NCSFIL` returns a nonzero value in `rcode`. Possible causes of errors include:

- The specified netCDF ID does not refer to an open netCDF file.
- The specified netCDF ID refers to a file open for read-only access.
- The fillmode argument is neither `NC_NOFILL` nor `NC_FILL` (neither `NCNOFILL` nor `NCFILL` for FORTRAN).

C Interface: ncsetfill

```
int ncsetfill(int ncid, int fillmode);
```

`ncid` NetCDF ID, returned from a previous call to `ncopen` or `nccreate`.

`fillmode` Desired fill mode for the file, either `NC_NOFILL` or `NC_FILL`.

`ncsetfill`
Returns the current fill mode of the file before this call, either `NC_NOFILL` or `NC_FILL`.

Here is an example using `ncsetfill` to set nofill mode for subsequent writes of a netCDF file named '`foo.nc`':

```
#include <netcdf.h>
   ...
int ncid;
   ...
ncid = ncopen("foo.nc", NC_WRITE);  /* open for writing */

   ...          /* write data with default prefilling behavior */

ncsetfill(ncid, NC_NOFILL);      /* set nofill mode */

   ...          /* write data with no prefilling */
```

FORTRAN Interface: NCSFIL

```
      INTEGER FUNCTION NCSFIL(INTEGER NCID, INTEGER FILLMODE,
     +                        INTEGER RCODE)
```

`NCID`	NetCDF ID, returned from a previous call to `NCOPN` or `NCCRE`.
`FILLMODE`	Desired fill mode for the file, either `NCNOFILL` or `NCFILL`.
`RCODE`	Returned error code. If no errors occurred, 0 is returned.
`NCSFIL`	Returns the current fill mode of the file before this call, either `NCNOFILL` or `NCFILL`.

Here is an example using `NCSFIL` to set nofill mode for a netCDF file named '`foo.nc`':

```
      INCLUDE 'netcdf.inc'
         ...
      INTEGER NCID, RCODE, OMODE
         ...
      NCID = NCOPN('foo.nc', NCWRITE, RCODE)
         ...
* write data with default prefilling behavior
         ...
      OMODE = NCSFIL(NCID, NCNOFILL, RCODE)
         ...
* write data with no prefilling
         ...
```

6 Dimensions

Dimensions for a netCDF file are defined when it is created, while the netCDF file is in define mode. Additional dimensions may be added later by reentering define mode. A netCDF dimension has a name and a size. At most one dimension in a netCDF file can have the `NC_UNLIMITED` size, which means variables using this dimension can grow along this dimension.

There is a suggested limit (100) to the number of dimensions that can be defined in a single netCDF file. The limit is the value of the predefined macro `MAX_NC_DIMS` (MAXNCDIM for FORTRAN). The purpose of the limit is to make writing generic applications simpler. They need only provide an array of `MAX_NC_DIMS` dimensions to handle any netCDF file. The implementation of the netCDF library does not enforce this advisory maximum, so it is possible to use more dimensions, if necessary; just don't expect generic applications or netCDF utilities to be able to handle the resulting netCDF files.

Ordinarily, the name and size of a dimension are fixed when the dimension is first defined. The name may be changed later, but the size of a dimension cannot be changed without copying all the data to a new netCDF file with a redefined dimension size.

Dimension sizes in the C interface are type `long` rather than type `int` to make it possible to access all the data in a netCDF file on a platform that only supports a 16-bit `int` data type, for example MSDOS. If dimension sizes were type `int` instead, it would not be possible to access data from variables with a dimension size greater than a 16-bit `int` can accommodate.

A netCDF dimension in an open netCDF file is referred to in the C and FORTRAN interfaces by a small integer called a *dimension ID*. In the C interface, dimension IDs are 0, 1, 2, ..., whereas in the FORTRAN interface, the associated IDs are instead 1, 2, 3, ..., in the order in which the dimensions were defined.

Operations supported on dimensions are:

- Create a dimension, given its name and size.
- Get a dimension ID from its name.
- Get a dimension's name and size from its ID.
- Rename a dimension.

6.1 Create a Dimension: ncdimdef and NCDDEF

The function `ncdimdef` (or `NCDDEF` for FORTRAN) adds a new dimension to an open netCDF file in define mode. It returns a dimension ID, given the netCDF ID, the dimension name, and the dimension size. At most one unlimited size dimension, called the record dimension, may be defined for each netCDF file.

In case of an error, `ncdimdef` returns -1; `NCDDEF` returns a nonzero value in `rcode`. Possible causes of errors include:

- The netCDF file is not in definition mode.
- The specified dimension name is the name of another existing dimension.
- The specified size is not greater than zero.
- The specified size is unlimited, but there is already an unlimited size dimension defined for this netCDF file.
- The specified netCDF ID does not refer to an open netCDF file.

C Interface: ncdimdef

```
int ncdimdef(int ncid, const char* name, long size);
```

`ncid`	NetCDF ID, returned from a previous call to `ncopen` or `nccreate`.
`name`	Dimension name. Must begin with an alphabetic character, followed by zero or more alphanumeric characters including the underscore ('`_`'). Case is significant.
`size`	Size of dimension; that is, number of values for this dimension as an index to variables that use it. This should be either a positive integer (of type `long`) or the predefined constant `NC_UNLIMITED`.

Here is an example using `ncdimdef` to create a dimension named `lat` of size 18 and a record dimension named `rec` in a new netCDF file named '`foo.nc`':

```
#include <netcdf.h>
   ...
int ncid, latid, recid;
   ...
ncid = nccreate("foo.nc", NC_NOCLOBBER);
   ...
latid = ncdimdef(ncid, "lat", 18L);
recid = ncdimdef(ncid, "rec", NC_UNLIMITED);
```

FORTRAN Interface: NCDDEF

```
      INTEGER FUNCTION NCDDEF (INTEGER NCID,
     +                    CHARACTER*(*) DIMNAM,
     +                    INTEGER DIMSIZ,
     +                    INTEGER RCODE)
```

`NCID`	NetCDF ID, returned from a previous call to `NCOPN` or `NCCRE`.
`DIMNAM`	Dimension name. Must begin with an alphabetic character, followed by zero or more alphanumeric characters including the underscore ('`_`'). Case is significant.
`DIMSIZ`	Size of dimension; that is, number of values for this dimension as an index to variables that use it. This should be either a positive integer or the predefined constant `NCUNLIM`.
`RCODE`	Returned error code. If no errors occurred, 0 is returned.

Here is an example using `NCDDEF` to create a dimension named `lat` of size 18 and a record dimension named `rec` in a new netCDF file named '`foo.nc`':

```
INCLUDE 'netcdf.inc'
   ...
INTEGER NCID, RCODE, LATID, RECID
   ...
NCID = NCCRE('foo.nc', NCNOCLOB, RCODE)
   ...
LATID = NCDDEF(NCID, 'lat', 18, RCODE)
RECID = NCDDEF(NCID, 'rec', NCUNLIM, RCODE)
```

6.2 Get a Dimension ID from Its Name: ncdimid and NCDID

The function `ncdimid` (or `NCDID` for FORTRAN) returns the ID of a netCDF dimension, given the name of the dimension. If `ndims` is the number of dimensions defined for a netCDF file, each dimension has an ID between `0` and `ndims-1` (or `1` and `ndims` for FORTRAN).

In case of an error, `ncdimid` returns -1; `NCDID` returns a nonzero value in `rcode`. Possible causes of errors include:

- The name that was specified is not the name of any currently defined dimension in the netCDF file.
- The specified netCDF ID does not refer to an open netCDF file.

C Interface: ncdimid

```
int ncdimid(int ncid, const char* name);
```

`ncid`	NetCDF ID, returned from a previous call to `ncopen` or `nccreate`.
`name`	Dimension name, a character string beginning with a letter and followed by any sequence of letters, digits, or underscore ('`_`') characters. Case is significant in dimension names.

Here is an example using `ncdimid` to determine the dimension ID of a dimension named `lat`, assumed to have been defined previously in an existing netCDF file named '`foo.nc`':

```
#include <netcdf.h>
   ...
int ncid, latid;
   ...
ncid = ncopen("foo.nc", NC_NOWRITE);  /* open for reading */
   ...
latid = ncdimid(ncid, "lat");
```

FORTRAN Interface: NCDID

```
      INTEGER FUNCTION NCDID (INTEGER NCID,
     +                        CHARACTER*(*) DIMNAME,
     +                        INTEGER RCODE)
```

`NCID` NetCDF ID, returned from a previous call to `NCOPN` or `NCCRE`.

`DIMNAME` Dimension name, a character string beginning with a letter and followed by any sequence of letters, digits, or underscore ('`_`') characters. Case is significant in dimension names.

`RCODE` Returned error code. If no errors occurred, 0 is returned.

Here is an example using `NCDID` to determine the dimension ID of a dimension named `lat`, assumed to have been defined previously in an existing netCDF file named '`foo.nc`':

```
      INCLUDE 'netcdf.inc'
         ...
      INTEGER NCID, RCODE, LATID
         ...
      NCID = NCOPN('foo.nc', NCNOWRIT, RCODE)
         ...
      LATID = NCDID(NCID, 'lat', RCODE)
```

6.3 Inquire about a Dimension: ncdiminq and NCDINQ

The function `ncdiminq` (or `NCDINQ` for FORTRAN) returns the name and size of a dimension, given its ID. The size for the unlimited dimension, if any, is the number of records written so far.

In case of an error, `ncdiminq` returns -1; `NCDINQ` returns a nonzero value in `rcode`. Possible causes of errors include:

- The dimension ID is invalid for the specified netCDF file.
- The specified netCDF ID does not refer to an open netCDF file.

C Interface: ncdiminq

```
    int ncdiminq(int ncid, int dimid, char* name, long* size);
```

`ncid` NetCDF ID, returned from a previous call to `ncopen` or `nccreate`.

`dimid` Dimension ID, as returned from a previous call to `ncdimid` or `ncdimdef`.

`name` Returned dimension name. The caller must allocate space for the returned name. The maximum possible length, in characters, of a dimension name is given by the predefined constant `MAX_NC_NAME`. If the name parameter is given as '`0`' (a null pointer), no name will be returned so no space needs to be allocated.

`size` Returned size of dimension. For the unlimited dimension, this is the number of records written so far. If this parameter is '`0`' (a null pointer), the size will not be returned, so no space for this information need be declared or allocated.

Here is an example using `ncdiminq` to determine the size of a dimension named `lat`, and the name and current maximum size of the unlimited (or record) dimension for an existing netCDF file named '`foo.nc`':

```
#include <netcdf.h>
   ...
int ncid, latid, ndims, nvars, ngatts, recid;
long latsize, recs;
char recname[MAX_NC_NAME];
   ...
ncid = ncopen("foo.nc", NC_NOWRITE);  /* open for reading */
   ...
latid = ncdimid(ncid, "lat");
/* get lat size, but don't get name, since we already know it */
ncdiminq(ncid, latid, 0, &latsize);
/* get ID of record dimension (among other things) */
ncinquire(ncid, &ndims, &nvars, &ngatts, &recid);
/* get record dimension name and current size */
ncdiminq(ncid, recid, recname, &recs);
```

FORTRAN Interface: NCDINQ

```
      SUBROUTINE NCDINQ (INTEGER NCID, INTEGER DIMID,
     +                   CHARACTER*(*) DIMNAM, INTEGER DIMSIZ,
     +                   INTEGER RCODE)
```

`NCID` NetCDF ID, returned from a previous call to `NCOPN` or `NCCRE`.

`DIMID` Dimension ID, as returned from a previous call to `NCDID` or `NCDDEF`.

`DIMNAM` Returned dimension name. The caller must allocate space for the returned name. The maximum possible length, in characters, of a dimension name is given by the predefined constant `MAXNCNAM`.

`DIMSIZ` Returned size of dimension. For the unlimited dimension, this is the current maximum value used for writing any variables with this dimension, that is the maximum record number.

`RCODE` Returned error code. If no errors occurred, 0 is returned.

Here is an example using `NCDINQ` to determine the size of a dimension named `lat`, and the name and current maximum size of the unlimited (or record) dimension for an existing netCDF file named '`foo.nc`':

```
      INCLUDE 'netcdf.inc'
         ...
      INTEGER NCID, RCODE, LATID, LATSIZ
      INTEGER NDIMS, NVARS, NGATTS, RECID, NRECS
* 31 in following statement is parameter MAXNCNAM
      CHARACTER*31 LATNAM, RECNAM
         ...
      NCID = NCOPN('foo.nc', NCNOWRIT, RCODE)
         ...
      LATID = NCDID(NCID, 'lat', RCODE)
* get lat name and size, (even though we already know name)
      CALL NCDINQ(NCID, LATID, LATNAM, LATSIZ, RCODE)
* get ID of record dimension (among other things)
      CALL NCINQ(NCID, NDIMS, NVARS, NGATTS, RECID, RCODE)
* get record dimension name and current size
      CALL NCDINQ(NCID, RECID, RECNAME, NRECS, RCODE)
```

6.4 Rename a Dimension: ncdimrename and NCDREN

The function `ncdimrename` (or `NCDREN` for FORTRAN) renames an existing dimension in a netCDF file open for writing. If the new name is longer than the old name, the netCDF dataset must be in define mode. You cannot rename a dimension to have the same name as another dimension.

In case of an error, `ncdimrename` returns -1; `NCDREN` returns a nonzero value in `rcode`. Possible causes of errors include:

- The new name is the name of another dimension.
- The dimension ID is invalid for the specified netCDF file.
- The specified netCDF ID does not refer to an open netCDF file.
- The new name is longer than the old name and the netCDF file is not in define mode.

C Interface: ncdimrename

```
int ncdimrename(int ncid, int dimid, const char* name);
```

`ncid`	NetCDF ID, returned from a previous call to `ncopen` or `nccreate`.
`dimid`	Dimension ID, as returned from a previous call to `ncdimid` or `ncdimdef`.
`name`	New dimension name.

Here is an example using `ncdimrename` to rename the dimension `lat` to `latitude` in an existing netCDF file named '`foo.nc`':

```
#include <netcdf.h>
   ...
int ncid, latid;
   ...
ncid = ncopen("foo.nc", NC_WRITE);  /* open for writing */
   ...
ncredef(ncid);  /* put in define mode to rename dimension */
latid = ncdimid(ncid, "lat");
ncdimrename(ncid, latid, "latitude");
ncendef(ncid);  /* leave define mode */
```

FORTRAN Interface: NCDREN

```
      SUBROUTINE NCDREN (INTEGER NCID, INTEGER DIMID,
     +                   CHARACTER*(*) DIMNAME, INTEGER RCODE)
```

`NCID`	NetCDF ID, returned from a previous call to `NCOPN` or `NCCRE`.
`DIMID`	Dimension ID, as returned from a previous call to `NCDID` or `NCDDEF`.
`DIMNAM`	New name for the dimension.
`RCODE`	Returned error code. If no errors occurred, 0 is returned.

Here is an example using `NCDREN` to rename the dimension "lat" to "latitude" in an existing netCDF file named '`foo.nc`':

```
      INCLUDE 'netcdf.inc'
         ...
      INTEGER NCID, RCODE, LATID
         ...
      NCID = NCOPN('foo.nc', NCWRITE, RCODE)
         ...
* put in define mode to rename dimension
      CALL NCREDF(NCID, RCODE)
      LATID = NCDID(NCID, 'lat', RCODE)
      CALL NCDREN(NCID, LATID, 'latitude', RCODE)
* leave define mode
      CALL NCENDF(NCID, RCODE)
```

7 Variables

Variables for a netCDF file are defined when the file is created, while the netCDF file is in define mode. Other variables may be added later by reentering define mode. A netCDF variable has a name, a type, and a shape, which are specified when it is defined. A variable may also have values, which are established later in data mode.

Ordinarily, the name, type, and shape are fixed when the variable is first defined. The name may be changed, but the type and shape of a variable cannot be changed. However, a variable defined in terms of the unlimited dimension can grow without bound in that dimension.

A netCDF variable in an open netCDF file is referred to in the C and FORTRAN interfaces by a small integer called a *variable ID*. Variable IDs reflect the order in which variables were defined within an netCDF file. In the C interface, variable IDs are 0, 1, 2, ..., whereas in the FORTRAN interface, they are instead 1, 2, 3, ..., in the order in which the variables were defined. A function is available in each interface for getting the variable ID from the variable name and vice-versa.

Attributes (see Chapter 8 [Attributes], page 101) may be associated with a variable to specify such properties as units.

Operations supported on variables are:

- Create a variable, given its name, data type, and shape.
- Get a variable ID from its name.
- Get a variable's name, data type, shape, and number of attributes from its ID.
- Put a data value into a variable, given variable ID, indices, and value.
- Put an array of values into a variable, given variable ID, corner indices, edge lengths, and a block of values.
- Put a subsampled or mapped array-section of values into a variable, given variable ID, corner indices, edge lengths, stride vector, index mapping vector, and a block of values.
- Put values into record variables, given record number and pointers to blocks of values.
- Get a data value from a variable, given variable ID and indices.
- Get an array of values from a variable, given variable ID, corner indices, and edge lengths.
- Get a subsampled or mapped array-section of values from a variable, given variable ID, corner indices, edge lengths, stride vector, and index mapping vector.
- Get values from record variables, given record number and pointers to where the data should be stored for each record variable.
- Rename a variable.
- Get number of bytes for a given data type.
- Get the number of record variables, their IDs, and their record sizes.

7.1 Language Types Corresponding to NetCDF Data Types

The following table gives the correspondence between netCDF data types and C and FORTRAN data types:

netCDF/	C		FORTRAN		
CDL Data	Data	API	Data	API	
Type	Type	Mnemonic	Type	Mnemonic	Bits
byte	char	NC_BYTE	BYTE, LOGICAL*1 (INTEGER)	NCBYTE	8
char	char	NC_CHAR	CHARACTER	NCCHAR	8
short	short	NC_SHORT	INTEGER*2 (INTEGER)	NCSHORT	16
long	nclong	NC_LONG	INTEGER*4 (INTEGER)	NCLONG	32
float	float	NC_FLOAT	REAL*4 (REAL)	NCFLOAT	32
double	double	NC_DOUBLE	REAL*8 (DOUBLEPRECISION)	NCDOUBLE	64

The first column gives the netCDF data type, which is the same as the CDL data type. The next pair of columns give, respectively, the C data type corresponding to the first column and the corresponding C preprocessor macro for use in netCDF functions (the preprocessor macros are defined in the netCDF C header-file `netcdf.h`). The next pair of columns give, respectively, the FORTRAN data type corresponding to the first column and the corresponding FORTRAN parameter for use when calling netCDF routines (the parameters are defined in the netCDF FORTRAN include-file `netcdf.inc`). You should use the un-parenthesized FORTRAN types if possible. For any type that your FORTRAN compiler doesn't support, use the corresponding parenthesized type. The last column gives the number of bits used in the external representation of values of the corresponding type.

Note that the C data type corresponding to a netCDF `long` is `nclong`. This type should be used rather than `int` or `long`. It is defined in the netCDF header-file `netcdf.h`, where it is set to the appropriate type.

Note that there are no netCDF types corresponding to 64-bit integers or to wide characters in the current version of the netCDF library.

7.2 Create a Variable: ncvardef and NCVDEF

The function `ncvardef` (or `NCVDEF` for FORTRAN) adds a new variable to an open netCDF file in define mode. It returns a variable ID, given the netCDF ID, the variable name, the variable type, the number of dimensions, and a list of the dimension IDs.

In case of an error, `ncvardef` returns -1; `NCVDEF` returns a nonzero value in `rcode`. Possible causes of errors include:

- The netCDF file is not in define mode.
- The specified variable name is the name of another existing variable.
- The specified type is not a valid netCDF type.

- The specified number of dimensions is negative or more than the constant `MAX_VAR_DIMS`, the maximum number of dimensions permitted for a netCDF variable.
- One or more of the dimension IDs in the list of dimensions is not a valid dimension ID for the netCDF file.
- The specified netCDF ID does not refer to an open netCDF file.

C Interface: ncvardef

```
int ncvardef(int ncid, const char* name, nc_type datatype,
             int ndims, const int dimids[]);
```

`ncid`	NetCDF ID, returned from a previous call to `ncopen` or `nccreate`.
`name`	Variable name. Must begin with an alphabetic character, followed by zero or more alphanumeric characters including the underscore ('`_`'). Case is significant.
`datatype`	One of the set of predefined netCDF data types. The type of this parameter, `nc_type`, is defined in the netCDF header file. The valid netCDF data types are `NC_BYTE`, `NC_CHAR`, `NC_SHORT`, `NC_LONG`, `NC_FLOAT`, and `NC_DOUBLE`.
`ndims`	Number of dimensions for the variable. For example, `2` specifies a matrix, `1` specifies a vector, and `0` means the variable is a scalar with no dimensions. Must not be negative or greater than the predefined constant `MAX_VAR_DIMS`.
`dimids`	Vector of `ndims` dimension IDs corresponding to the variable dimensions. If the ID of the unlimited dimension is included, it must be first. This argument is ignored if `ndims` is `0`.

Here is an example using `ncvardef` to create a variable named `rh` of type `long` with three dimensions, `time`, `lat`, and `lon` in a new netCDF file named '`foo.nc`':

```
#include <netcdf.h>
   ...
int  ncid;                          /* netCDF ID */
int  lat_dim, lon_dim, time_dim;    /* dimension IDs */
int  rh_id;                         /* variable ID */
int  rh_dimids[3];                  /* variable shape */
   ...
ncid = nccreate("foo.nc", NC_CLOBBER);
   ...
                                    /* define dimensions */
lat_dim = ncdimdef(ncid, "lat", 5L);
lon_dim = ncdimdef(ncid, "lon", 10L);
time_dim = ncdimdef(ncid, "time", NC_UNLIMITED);
   ...
                                   /* define variable */
rh_dimids[0] = time_dim;
rh_dimids[1] = lat_dim;
rh_dimids[2] = lon_dim;
rh_id = ncvardef (ncid, "rh", NC_DOUBLE, 3, rh_dimids);
```

FORTRAN Interface: NCVDEF

```
      INTEGER FUNCTION NCVDEF(INTEGER NCID, CHARACTER*(*) VARNAM,
     +                        INTEGER VARTYP, INTEGER NVDIMS,
     +                        INTEGER VDIMS(*), INTEGER RCODE)
```

`NCID` NetCDF ID, returned from a previous call to `NCOPN` or `NCCRE`.

`VARNAM` Variable name. Must begin with an alphabetic character, which is followed by zero or more alphanumeric characters including the underscore ('`_`'). Case is significant.

`VARTYP` One of the set of predefined netCDF data types. The valid netCDF data types are `NCBYTE`, `NCCHAR`, `NCSHORT`, `NCLONG`, `NCFLOAT`, and `NCDOUBLE`.

`NVDIMS` Number of dimensions for the variable. For example, `2` specifies a matrix, `1` specifies a vector, and `0` means the variable is a scalar with no dimensions. Must not be negative or greater than the predefined constant `MAXVDIMS`.

`VDIMS` Vector of `NVDIMS` dimension IDs corresponding to the variable dimensions. If the ID of the unlimited dimension is included, it must be last. This argument is ignored if `NVDIMS` is `0`.

`RCODE` Returned error code. If no errors occurred, 0 is returned.

Here is an example using `NCVDEF` to create a variable named `rh` of type `long` with three dimensions, `time`, `lat`, and `lon` in a new netCDF file named '`foo.nc`':

```
INCLUDE 'netcdf.inc'
   ...
INTEGER  NCID, RCODE
INTEGER  LATDIM, LONDIM, TIMDIM  ! dimension IDs
INTEGER  RHID                    ! variable ID
INTEGER  RHDIMS(3)               ! variable shape
   ...
NCID = NCCRE ('foo.nc', NC_CLOBBER, RCODE)
   ...
                                 ! define dimensions
LATDIM = NCDDEF(NCID, 'lat', 5, RCODE)
LONDIM = NCDDEF(NCID, 'lon', 10, RCODE)
TIMDIM = NCDDEF(NCID, 'time', NCUNLIM, RCODE)
   ...
                                 ! define variable
RHDIMS(1) = LONDIM
RHDIMS(2) = LATDIM
RHDIMS(3) = TIMDIM
RHID = NCVDEF (NCID, 'rh', NCDOUBLE, 3, RHDIMS, RCODE)
```

7.3 Get a Variable ID from Its Name: ncvarid and NCVID

The function `ncvarid` (or `NCVID` for FORTRAN) returns the ID of a netCDF variable, given its name.

In case of an error, `ncvarid` returns -1; `NCVID` returns a nonzero value in `rcode`. Possible causes of errors include:

- The specified variable name is not a valid name for a variable in the specified netCDF file.
- The specified netCDF ID does not refer to an open netCDF file.

C Interface: ncvarid

```
int ncvarid(int ncid, const char* name);
```

`ncid`	NetCDF ID, returned from a previous call to `ncopen` or `nccreate`.
`name`	Variable name for which ID is desired.

Here is an example using `ncvarid` to find out the ID of a variable named `rh` in an existing netCDF file named '`foo.nc`':

```
#include <netcdf.h>
   ...
int  ncid;                       /* netCDF ID */
int  rh_id;                      /* variable ID */
   ...
ncid = ncopen("foo.nc", NC_NOWRITE);
   ...
rh_id = ncvarid (ncid, "rh");
```

FORTRAN Interface: NCVID

```
      INTEGER FUNCTION NCVID(INTEGER NCID,
     +                       CHARACTER*(*) VARNAM,
     +                       INTEGER RCODE)
```

`NCID`	NetCDF ID, returned from a previous call to `NCOPN` or `NCCRE`.
`VARNAM`	Variable name for which ID is desired.
`RCODE`	Returned error code. If no errors occurred, 0 is returned.

Here is an example using `NCVID` to find out the ID of a variable named `rh` in an existing netCDF file named '`foo.nc`':

```
      INCLUDE 'netcdf.inc'
         ...
      INTEGER  NCID, RCODE
      INTEGER  RHID                    ! variable ID
         ...
      NCID = NCOPN ('foo.nc', NCNOWRIT, RCODE)
         ...
      RHID = NCVID (NCID, 'rh', RCODE)
```

7.4 Get Information about a Variable from Its ID: ncvarinq and NCVINQ

The function `ncvarinq` (or `NCVINQ` for FORTRAN) returns information about a netCDF variable, given its ID. The information returned is the name, type, number of dimensions, a list of dimension IDs describing the shape of the variable, and the number of variable attributes that have been assigned to the variable.

In case of an error, `ncvarinq` returns -1; `NCVINQ` returns a nonzero value in `rcode`. Possible causes of errors include:

- The variable ID is invalid for the specified netCDF file.
- The specified netCDF ID does not refer to an open netCDF file.

C Interface: ncvarinq

```
int ncvarinq(int ncid, int varid, char* name, nc_type* datatype,
             int* ndims, int dimids[], int* natts);
```

`ncid` NetCDF ID, returned from a previous call to `ncopen` or `nccreate`.

`varid` Variable ID.

`name` Returned variable name. The caller must allocate space for the returned name. The maximum possible length, in characters, of a variable name is given by the predefined constant `MAX_NC_NAME`. If the name parameter is given as '`0`' (a null pointer), no name will be returned so no space needs to be allocated.

`datatype` Returned variable type, one of the set of predefined netCDF data types. The type of this parameter, `nc_type`, is defined in the netCDF header file. The valid netCDF data types are `NC_BYTE`, `NC_CHAR`, `NC_SHORT`, `NC_LONG`, `NC_FLOAT`, and `NC_DOUBLE`. If this parameter is given as '`0`' (a null pointer), no type will be returned so no variable to hold the type needs to be declared.

`ndims` Returned number of dimensions the variable was defined as using. For example, `2` specifies a matrix, `1` specifies a vector, and `0` means the variable is a scalar with no dimensions. If this parameter is given as '`0`' (a null pointer), no number of dimensions will be returned so no variable to hold this information needs to be declared.

`dimids` Returned vector of `ndims` dimension IDs corresponding to the variable dimensions. The caller must allocate enough space for a vector of at least `ndims` integers to be returned. The maximum possible number of dimensions for a variable is given by the predefined constant `MAX_VAR_DIMS`. If this parameter is given as '`0`' (a null pointer), no vector will be returned so no space to hold the dimension IDs needs to be declared or allocated.

`natts` Returned number of variable attributes assigned to this variable. If this parameter is given as '`0`' (a null pointer), the number of attributes will not be returned so no space to hold this information needs to be declared or allocated.

Here is an example using `ncvarinq` to find out about a variable named `rh` in an existing netCDF file named '`foo.nc`':

```
#include <netcdf.h>
   ...
int  ncid;                         /* netCDF ID */
int  rh_id;                        /* variable ID */
nc_type rh_type;                   /* variable type */
int rh_ndims;                      /* number of dims */
int  rh_dims[MAX_VAR_DIMS];        /* variable shape */
int rh_natts                       /* number of attributes */
   ...
ncid = ncopen ("foo.nc", NC_NOWRITE);
   ...
rh_id = ncvarid (ncid, "rh");
/* we don't need name, since we already know it */
ncvarinq (ncid, rh_id, 0, &rh_type, &rh_ndims, rh_dims, &rh_natts);
```

FORTRAN Interface: NCVINQ

```
      SUBROUTINE NCVINQ (INTEGER NCID, INTEGER VARID,
     +                   CHARACTER*(*) VARNAM, INTEGER VARTYP,
     +                   INTEGER NVDIMS, INTEGER VDIMS(*),
     +                   INTEGER NVATTS, INTEGER RCODE)
```

`NCID`	NetCDF ID, returned from a previous call to `NCOPN` or `NCCRE`.
`VARID`	Variable ID.
`VARNAM`	Returned variable name. The caller must allocate space for the returned name. The maximum possible length, in characters, of a variable name is given by the predefined constant `MAXNCNAM`.
`VARTYP`	Returned variable type, one of the set of predefined netCDF data types. The valid netCDF data types are `NCBYTE`, `NCCHAR`, `NCSHORT`, `NCLONG`, `NCFLOAT`, and `NCDOUBLE`.
`NVDIMS`	Returned number of dimensions for the variable. For example, `2` specifies a matrix, `1` specifies a vector, and `0` means the variable is a scalar with no dimensions.
`VDIMS`	Returned vector of `NVDIMS` dimension IDs corresponding to the variable dimensions. The caller must allocate enough space for a vector of at least `NVDIMS` integers to be returned. The maximum possible number of dimensions for a variable is given by the predefined constant `MAXVDIMS`.
`NVATTS`	Returned number of variable attributes assigned to this variable.
`RCODE`	Returned error code. If no errors occurred, 0 is returned.

Here is an example using `NCVINQ` to find out about a variable named `rh` in an existing netCDF file named '`foo.nc`':

```
      INCLUDE 'netcdf.inc'
         ...
      INTEGER  NCID, RCODE
      INTEGER  RHID                  ! variable ID
      CHARACTER*31 RHNAME            ! variable name
      INTEGER  RHTYPE                ! variable type
      INTEGER  RHN                   ! number of dimensions
      INTEGER  RHDIMS(MAXVDIMS)      ! variable shape
      INTEGER  RHNATT                ! number of attributes
         ...
      NCID = NCOPN ('foo.nc', NCNOWRIT, RCODE)
         ...
      RHID = NCVID (NCID, 'rh', RCODE)  ! get ID
      CALL NCVINQ (NCID, RHID, RHNAME, RHTYPE, RHN, RHDIMS, RHNATT,
     +             RCODE)
```

7.5 Write a Single Data Value: ncvarput1, NCVPT1, and NCVP1C

The function `ncvarput1` (or `NCVPT1` or `NCVP1C` for FORTRAN) puts a single data value into a variable of an open netCDF file that is in data mode. Inputs are the netCDF ID, the variable ID, a multidimensional index that specifies which value to add or alter, and the data value.

In case of an error, `ncvarput1` returns -1; `NCVPT1` returns a nonzero value in `rcode`. Possible causes of errors include:

- The variable ID is invalid for the specified netCDF file.
- The specified indices were out of range for the rank of the specified variable. For example, a negative index or an index that is larger than the corresponding dimension size will cause an error.
- The specified netCDF is in define mode rather than data mode.
- The specified netCDF ID does not refer to an open netCDF file.

C Interface: ncvarput1

```
int ncvarput1(int ncid, int varid, const long mindex[], const void *value);
```

`ncid` NetCDF ID, returned from a previous call to `ncopen` or `nccreate`.

`varid` Variable ID.

`mindex` The multidimensional index of the the data value to be written. The indices are relative to 0, so for example, the first data value of a two-dimensional variable would have index `(0,0)`. The elements of `mindex` must correspond to the variable's dimensions. Hence, if the variable is a record variable, the first index would correspond to the record number.

`value` Pointer to the data value to be written. The pointer is declared to be of type `void *` because it can point to data of any of the basic netCDF types. The data should be of the appropriate type for the netCDF variable. **Warning: neither the compiler nor the netCDF software can detect whether the wrong type of data is used.**

Here is an example using `ncvarput1` to set the `(1,2,3)` element of the variable named `rh` to `0.5` in an existing netCDF file named '`foo.nc`'. For simplicity in this example, we assume that we know that `rh` is dimensioned with `time`, `lat`, and `lon`, so we want to set the value of `rh` that corresponds to the second `time` value, the third `lat` value, and the fourth `lon` value:

```
#include <netcdf.h>
   ...
int  ncid;                         /* netCDF ID */
int  rh_id;                        /* variable ID */
static long rh_index[] = {1, 2, 3}; /* where to put value */
static double rh_val = 0.5;        /* value to put */
   ...
ncid = ncopen("foo.nc", NC_WRITE);
   ...
rh_id = ncvarid (ncid, "rh");
   ...
ncvarput1(ncid, rh_id, rh_index, &rh_val);
```

FORTRAN Interface: NCVPT1

```
 SUBROUTINE NCVPT1 (INTEGER NCID, INTEGER VARID,
+                   INTEGER MINDEX(*), type VALUE,
+                   INTEGER RCODE)

 SUBROUTINE NCVP1C (INTEGER NCID, INTEGER VARID,
+                   INTEGER MINDEX(*), CHARACTER CHVAL,
+                   INTEGER RCODE)
```

There are two FORTRAN subroutines, `NCVPT1` and `NCVP1C`, for putting a single value in a variable. The first puts a numeric value in a variable of numeric type, and the second puts a character value in a variable of character type.

`NCID`	NetCDF ID, returned from a previous call to `NCOPN` or `NCCRE`.
`VARID`	Variable ID.
`MINDEX`	The multidimensional index of the the data value to be written. The indices are relative to 1, so for example, the first data value of a two-dimensional variable would have index `(1,1)`. The elements of `mindex` must correspond to the variable's dimensions. Hence, if the variable is a record variable, the last index would correspond to the record number.
`VALUE`	For `NCVPT1`, the data value to be written. The data may be of a type corresponding to any of the netCDF types `NCSHORT`, `NCLONG`, `NCFLOAT`, or `NCDOUBLE`, but must be appropriate for the type of the netCDF variable. **Warning: neither the compiler nor the netCDF software can detect whether the wrong type of data is used.**

`CHVAL`	For `NCVP1C`, the data value to be written. The data should be of a type character, corresponding to the netCDF types `NCCHAR` or `NCBYTE`.
`RCODE`	Returned error code. If no errors occurred, 0 is returned.

Here is an example using `NCVPT1` to set the `(4,3,2)` element of the variable named `rh` to `0.5` in an existing netCDF file named '`foo.nc`'. For simplicity in this example, we assume that we know that `rh` is dimensioned with `lon`, `lat`, and `time`, so we want to set the value of `rh` that corresponds to the fourth `lon` value, the third `lat` value, and the second `time` value:

```
INCLUDE 'netcdf.inc'
   ...
INTEGER  NCID, RCODE
INTEGER  RHID              ! variable ID
INTEGER  RHINDX(3)         ! where to put value
DATA RHINDX /4, 3, 2/
   ...
NCID = NCOPN ('foo.nc', NCWRITE, RCODE)
   ...
RHID = NCVID (NCID, 'rh', RCODE)  ! get ID
CALL NCVPT1 (NCID, RHID, RHINDX, 0.5, RCODE)
```

7.6 Write an Array of Values: ncvarput and NCVPT(C)

The function `ncvarput` (or `NCVPT` or `NCVPTC` for FORTRAN) writes values into a netCDF variable of an open netCDF file. The part of the netCDF variable to write is specified by giving a corner and a vector of edge lengths that refer to an array section of the netCDF variable. The values to be written are associated with the netCDF variable by assuming that the last dimension of the netCDF variable varies fastest in the C interface, whereas the first dimension of the netCDF variable varies fastest in the FORTRAN interface. The netCDF file must be in data mode. [1]

In case of an error, `ncvarput` returns -1; `NCVPT` returns a nonzero value in `rcode`. Possible causes of errors include:

- The variable ID is invalid for the specified netCDF file.
- The specified corner indices were out of range for the rank of the specified variable. For example, a negative index, or an index that is larger than the corresponding dimension size will cause an error.
- The specified edge lengths added to the specified corner would have referenced data out of range for the rank of the specified variable. For example, an edge length that is larger than the corresponding dimension size minus the corner index will cause an error.
- The specified netCDF file is in define mode rather than data mode.

[1] The current implementation of XDR on MSDOS systems restricts the amount of data accessed to no more than 64 Kbytes for each call to `ncvarput` (or `NCVPT` or `NCVPTC` for FORTRAN).

- The specified netCDF ID does not refer to an open netCDF file.

C Interface: ncvarput

```
int ncvarput(int ncid, int varid, const long start[], const long count[],
             const void *values);
```

`ncid` NetCDF ID, returned from a previous call to `ncopen` or `nccreate`.

`varid` Variable ID.

`start` A vector of long integers specifying the index in the variable where the first of the data values will be written. The indices are relative to 0, so for example, the first data value of a variable would have index `(0, 0, ..., 0)`. The size of `start` must be the same as the number of dimensions of the specified variable. The elements of `start` must correspond to the variable's dimensions in order. Hence, if the variable is a record variable, the first index would correspond to the starting record number for writing the data values.

`count` A vector of long integers specifying the edge lengths along each dimension of the block of data values to be written. To write a single value, for example, specify `count` as `(1, 1, ..., 1)`. The size of `count` is the number of dimensions of the specified variable. The elements of `count` correspond to the variable's dimensions. Hence, if the variable is a record variable, the first element of `count` corresponds to a count of the number of records to write.

`value` Pointer to a block of data values to be written. The order in which the data will be written to the netCDF variable is with the last dimension of the specified variable varying fastest. The pointer is declared to be of the type `void *` because it can point to data of any of the basic netCDF types. The data should be of the appropriate type for the netCDF variable. **Warning: neither the compiler nor the netCDF software can detect whether the wrong type of data is used.**

Here is an example using `ncvarput` to add or change all the values of the variable named `rh` to `0.5` in an existing netCDF file named '`foo.nc`'. For simplicity in this example, we assume that we know that `rh` is dimensioned with `time`, `lat`, and `lon`, and that there are three `time` values, five `lat` values, and ten `lon` values.

```
#include <netcdf.h>
   ...
#define TIMES 3
#define LATS  5
#define LONS  10
int  ncid;                         /* netCDF ID */
int  rh_id;                        /* variable ID */
static long start[] = {0, 0, 0};    /* start at first value */
static long count[] = {TIMES, LATS, LONS};
double rh_vals[TIMES*LATS*LONS];   /* array to hold values */
int i;
   ...
ncid = ncopen("foo.nc", NC_WRITE);
   ...
rh_id = ncvarid (ncid, "rh");
   ...
for (i = 0; i < TIMES*LATS*LONS; i++)
    rh_vals[i] = 0.5;
/* write values into netCDF variable */
ncvarput(ncid, rh_id, start, count, rh_vals);
```

FORTRAN Interface: NCVPT

```
      SUBROUTINE NCVPT (INTEGER NCID, INTEGER VARID,
     +                  INTEGER START(*), INTEGER COUNT(*),
     +                  type VALUES, INTEGER RCODE)

      SUBROUTINE NCVPTC(INTEGER NCID, INTEGER VARID,
     +                  INTEGER START(*), INTEGER COUNTS(*),
     +                  CHARACTER*(*) STRING, INTEGER LENSTR,
     +                  INTEGER RCODE)
```

There are two FORTRAN subroutines, `NCVPT` and `NCVPTC`, for writing an array of values into a netCDF variable. The first writes numeric values into a variable of numeric type, and the second writes character values into a variable of character type.

`NCID` NetCDF ID, returned from a previous call to `NCOPN` or `NCCRE`.

`VARID` Variable ID.

`START` A vector of integers specifying the index in the variable where the first of the data values will be written. The indices are relative to 1, so for example, the first data value of a variable would have index `(1, 1, ..., 1)`. The size of `START` must be the same as the number of dimensions of the specified variable. The elements of `START` must correspond to the variable's dimensions in order. Hence, if the variable is a record variable, the last index would correspond to the starting record number for writing the data values.

`COUNT`	A vector of integers specifying the edge lengths along each dimension of the block of data values to written. To write a single value, for example, specify `COUNT` as `(1, 1, ..., 1)`. The size of `COUNT` is the number of dimensions of the specified variable. The elements of `COUNT` correspond to the variable's dimensions. Hence, if the variable is a record variable, the last element of `COUNT` corresponds to a count of the number of records to write.
`VALUES`	For `NCVPT`, the block of data values to be written. The order in which the data will be written into the specified variable is with the first dimension varying fastest (like the ordinary FORTRAN convention). The data may be of a type corresponding to any of the netCDF types `NCSHORT`, `NCLONG`, `NCFLOAT`, or `NCDOUBLE`, but must be appropriate for the type of the netCDF variable. **Warning: neither the compiler nor the netCDF software can detect whether the wrong type of data is used.**
`STRING`	For `NCVPTC`, the characters to be written. The order in which the characters will be written to the netCDF variable is with the first dimension of the specified variable varying fastest (like the FORTRAN convention). The data may be of a type corresponding to the netCDF types `NCCHAR` or `NCBYTE`.
`LENSTR`	For `NCVPTC`, the total declared length (in characters) of the `STRING` argument. This should be at least as large as the product of the elements of the `COUNT` vector. Note that this is not necessarily the same as the value returned by the FORTRAN `LEN` function, because an array argument may be provided.
`RCODE`	Returned error code. If no errors occurred, 0 is returned.

Here is an example using `NCVPT` to add or change all the values of the variable named `rh` to `0.5` in an existing netCDF file named '`foo.nc`'. For simplicity in this example, we assume that we know that `rh` is dimensioned with `lon`, `lat`, and `time`, and that there are ten `lon` values, five `lat` values, and three `time` values.

```
      INCLUDE 'netcdf.inc'
         ...
      PARAMETER (NDIMS=3)         ! number of dimensions
      PARAMETER (TIMES=3, LATS=5, LONS=10) ! dimension sizes
      INTEGER  NCID, RCODE, TIMES
      INTEGER  RHID               ! variable ID
      INTEGER  START(NDIMS), COUNT(NDIMS)
      DOUBLE RHVALS(LONS, LATS, TIMES)
      DATA START /1, 1, 1/        ! start at first value
      DATA COUNT /LONS, LATS, TIMES/
         ...
      NCID = NCOPN ('foo.nc', NCWRITE, RCODE)
         ...
      RHID = NCVID (NCID, 'rh', RCODE)   ! get ID
      DO 10 ILON = 1, LONS
         DO 10 ILAT = 1, LATS
            DO 10 ITIME = 1, TIMES
               RHVALS(ILON, ILAT, ITIME) = 0.5
   10 CONTINUE
      CALL NCVPT (NCID, RHID, START, COUNT, RHVALS, RCODE)
```

7.7 Write a Subsampled Or Mapped Array of Values: ncvarputg, NCVPTG, and NCVPGC

The function `ncvarputg` (or `NCVPTG` or `NCVPGC` for FORTRAN) writes a subsampled or mapped array section of values into a netCDF variable of an open netCDF file. The subsampled or mapped array section is specified by giving a corner, a vector of edge lengths, a stride vector, and an index mapping vector. No assumptions are made about the ordering or size of the dimensions of the data array. The netCDF file must be in data mode.

In case of an error, `ncvarputg` returns -1; `NCVPTG` and `NCVPGC` return a nonzero value in `rcode`. Possible causes of errors include:

- The variable ID is invalid for the specified netCDF file.
- The specified corner indices were out of range for the rank of the specified variable. For example, a negative index, or an index that is larger than the corresponding dimension size will cause an error.
- The specified edge lengths and strides added to the specified corner would have referenced data out of range for the rank of the specified variable. For example, an edge length that is larger than the corresponding dimension size minus the corner index will cause an error, as will accessing two or more points using a stride that is greater than the size of the netCDF variable in the corresponding dimension.
- A non-positive stride.
- The specified netCDF is in define mode rather than data mode.

- The specified netCDF ID does not refer to an open netCDF file.

C Interface: ncvarputg

```
int ncvarputg(int ncid, int varid, const long start[], const long count[],
              const long stride[], const long imap[], const void *values);
```

`ncid` NetCDF ID, returned from a previous call to `ncopen` or `nccreate`.

`varid` Variable ID.

`start` A vector of long integers specifying the index in the variable where the first of the data values will be written. The indices are relative to 0, so for example, the first data value of a variable would have index `(0, 0, ..., 0)`. The elements of `start` correspond, in order, to the variable's dimensions. Hence, if the variable is a record variable, the first index corresponds to the starting record number for writing the data values.

`count` A vector of long integers specifying the edge lengths along each dimension of the block of data values to be written. To write a single value, for example, specify `count` as `(1, 1, ..., 1)`. The elements of `count` correspond, in order, to the variable's dimensions. Hence, if the variable is a record variable, the first element of `count` corresponds to a count of the number of records to write.

`stride` A vector of long integers specifying, for each dimension, the interval between the accessed values of a netCDF variable. The elements of the stride vector correspond, in order, to the variable's dimensions. A value of 1 accesses adjacent values of the netCDF variable in the corresponding dimension; a value of 2 accesses every other value of the netCDF variable in the corresponding dimension; and so on. A `NULL` stride argument obtains the default behavior in which adjacent values are accessed along each dimension.

`imap` A vector of long integers specifying, for each dimension, how data values associated with a netCDF variable are arranged in memory. The offset, in bytes, from the memory location pointed to by the `value` argument to a particular datum is given by the inner product of the index mapping vector with the coordinates of the datum. (The *inner product* of two vectors [x0, x1, ..., xn] and [y0, y1, ..., yn] is just x0*y0 + x1*y1 + ... + xn*yn.) The vector may contain negative values if the `value` argument is appropriately specified. A `NULL` argument obtains the default behavior in which the memory-resident values are assumed to have the same structure as the associated netCDF variable.

`value` Pointer to a block of data values to be written. The order in which the data will be written to the netCDF variable is with the last dimension of the netCDF variable varying fastest. The pointer is declared to be of the type `void *` because it can point to data of any of the basic netCDF types. The data should be of the appropriate type for the netCDF variable. **Warning: neither the compiler nor the netCDF software can detect whether the wrong type of data is used.**

Here is an example using `ncvarputg` to add or change every other value in each dimension of the variable named `rh` to `0.5` in an existing netCDF file named '`foo.nc`'. Values are taken, using the same dimensional strides, from points in a 3-dimensional array of structures whose dimensions are the reverse of the netCDF variable. For simplicity in this example, we assume that we know that `rh` is dimensioned with `time`, `lat`, and `lon`, and that there are three `time` values, five `lat` values, and ten `lon` values.

```
#include <netcdf.h>
   ...
#define TIMES 3
#define LATS  5
#define LONS  10
int  ncid;                           /* netCDF ID */
int  rh_id;                          /* variable ID */
static long start[] = {0, 0, 0};     /* start at first value */
static long count[] = {TIMES, LATS, LONS};
static long stride[] = {2, 2, 2};    /* every other value */
long imap[3];                        /* set to reverse of variable */
struct datum {
    int    dummy;                    /* to illustrate mapping vector */
    double rh_val;                   /* actual value to be written */
}      data[LONS][LATS][TIMES];      /* reversed array to hold values. */
int itime, ilat, ilon;
   ...
ncid = ncopen("foo.nc", NC_WRITE);
   ...
rh_id = ncvarid (ncid, "rh");
   ...
for (ilon = 0; ilon < LONS; ilon += stride[2])
    for (ilat = 0; ilat < LATS; ilat += stride[1])
        for (itime = 0; itime < TIMES; itime += stride[0])
            data[ilon][ilat][itime].rh_val = 0.5;
/* access every 'stride' in-memory value using reversed dimensions */
imap[0] = stride[2]*sizeof(struct datum);
imap[1] = stride[1]*(1+(LONS-1)/stride[0])*imap[0];
imap[2] = stride[0]*(1+(LATS-1)/stride[1])*imap[1];
/* write subsampled or mapped array of values into netCDF variable */
ncvarputg(ncid, rh_id, start, count, stride, imap, &data[0][0][0].rh_val);
```

FORTRAN Interface: NCVPTG, NCVPGC

```
      SUBROUTINE NCVPTG (INTEGER NCID, INTEGER VARID,
     +                   INTEGER START(*), INTEGER COUNT(*),
     +                   INTEGER STRIDE(*), INTEGER IMAP(*),
     +                   type VALUES, INTEGER RCODE)

      SUBROUTINE NCVPGC (INTEGER NCID, INTEGER VARID,
     +                   INTEGER START(*), INTEGER COUNT(*),
     +                    INTEGER STRIDE(*), INTEGER IMAP(*),
     +                    CHARACTER*(*) STRING, INTEGER RCODE)
```

There are two FORTRAN subroutines, `NCVPTG` and `NCVPGC`, for writing a subsampled or mapped array section of values into a netCDF variable. The first writes numeric values into a variable of numeric type, and the second writes character values into a variable of character type.

`NCID` NetCDF ID, returned from a previous call to `NCOPN` or `NCCRE`.

`VARID` Variable ID.

`START` A vector of integers specifying the index in the variable where the first of the data values will be written. The indices are relative to 1, so for example, the first data value of a variable would have index `(1, 1, ..., 1)`. The elements of `START` correspond, in order, to the variable's dimensions. Hence, if the variable is a record variable, the last index would correspond to the starting record number for writing the data values.

`COUNT` A vector of integers specifying the edge lengths along each dimension of the block of data values to be written. To write a single value, for example, specify `COUNT` as `(1, 1, ..., 1)`. The elements of `COUNT` correspond, in order, to the variable's dimensions. Hence, if the variable is a record variable, the last element of `COUNT` corresponds to a count of the number of records to write.

`STRIDE` A vector of integers specifying, for each dimension, the interval between the accessed values of a netCDF variable or the value `0`. The elements of the vector correspond, in order, to the variable's dimensions. A value of 1 accesses adjacent values of the netCDF variable in the corresponding dimension; a value of 2 accesses every other value of the netCDF variable in the corresponding dimension; and so on. An `0` argument obtains the default behavior in which adjacent values are accessed along each dimension.

`IMAP` A vector of integers specifying, for each dimension, how data values associated with a netCDF variable are arranged in memory or the value `0`. The offset, in bytes, from the memory location pointed to by the `value` argument to a particular datum is given by the inner product of the index mapping vector with the (origin-0) coordinates of the datum. (The *inner product* of two vectors [x1, x2, ..., xn] and [y1, y2, ..., yn] is just x1*y1 + x2*y2 + ... + xn*yn.) The vector may contain negative values if the `value` argument is appropriately specified. A `0` argument obtains the default behavior in which the memory-resident values are assumed to have the same structure as the associated netCDF variable.

`VALUES` For `NCVPTG`, the block of data values to be written. The order in which the data will be written is with the first dimension of the netCDF variable varying fastest (like the ordinary FORTRAN convention). The data may be of a type corresponding to any of the netCDF types `NCSHORT`, `NCLONG`, `NCFLOAT`, or `NCDOUBLE`, but must be appropriate for the type of the netCDF variable. **Warning: neither the compiler nor the netCDF software can detect whether the wrong type of data is used.**

`STRING` For `NCVPGC`, the characters to be written. The order in which the characters will be written to the netCDF variable is with the first dimension of the subsampled or mapped array varying fastest (like the FORTRAN convention). The data may be of a type corresponding to the netCDF types `NCCHAR` or `NCBYTE`.

`RCODE` Returned error code. If no errors occurred, 0 is returned.

Here is an example using `NCVPTG` to add or change every other value in each dimension of the variable named `rh` to `0.5` in an existing netCDF file named '`foo.nc`'. Values are taken, using the same dimensional strides, from a 2-parameter array whose dimensions are the reverse of the netCDF variable. For simplicity in this example, we assume that we know that `rh` is dimensioned with `lon`, `lat`, and `time`, and that there are ten `lon` values, five `lat` values, and three `time` values.

```
      INCLUDE 'netcdf.inc'
         ...
      PARAMETER (NDIMS=3)         ! number of dimensions
      PARAMETER (TIMES=3, LATS=5, LONS=10) ! dimension sizes
      INTEGER  NCID, RCODE
      INTEGER  RHID               ! variable ID
      INTEGER  START(NDIMS), COUNT(NDIMS),
     +         STRIDE(NDIMS), IMAP(NDIMS)  ! subsampled or mapped array
      DOUBLE DATA(2, TIMES, LATS, LONS)    ! rh is second parameter
      DATA START /1, 1, 1/        ! start at first value
      DATA COUNT /LONS, LATS, TIMES/
      DATA STRIDE /2, 2, 2/
         ...
      NCID = NCOPN ('foo.nc', NCWRITE, RCODE)
         ...
      RHID = NCVID (NCID, 'rh', RCODE)   ! get ID
      DO 10 ILON = 1, LONGS, STRIDE(1)
         DO 10 ILAT = 1, LATS, STRIDE(2)
            DO 10 ITIME = 1, TIMES, STRIDE(3)
               DATA(2, ITIME, ILAT, ILON) = 0.5
   10 CONTINUE
      IMAP(3) = 8*2*2   ! every other point of vector of 2-doubles
      IMAP(2) = IMAP(3)*(1+(TIMES-1)/STRIDE(3))*2
      IMAP(1) = IMAP(2)*(1+(LATS-1)/STRIDE(2))*2
      CALL NCVPTG (NCID, RHID, START, COUNT, STRIDE, IMAP,
     +            DATA(2,1,1,1), RCODE)
```

7.8 Put a Record: ncrecput

The function `ncrecput` writes a multi-variable record of values (or part of a record of values) into the record variables of an open netCDF file. The record is specified by giving a (0-based) record number. The values to be written are specified by an array of pointers, one for each record variable, to blocks of values. Each block of values should be of the appropriate size and type for a record's worth of data for the corresponding record variable. Each such pointer must be either

'`0`' (a null pointer), to indicate that no data is to be written for that variable, or must point to an entire record's worth of data of the appropriate type for the corresponding record variable. The values for each record variable are assumed to be ordered with the last dimension varying fastest. The netCDF file must be in data mode.

The `ncrecput` function is not strictly necessary, since the same data may be written with a sequence of calls to `ncvarput`, one for each record variable for which a non-null pointer is specified. This function is provided in the C interface for convenience only; no corresponding FORTRAN interface is available, so FORTRAN users should use multiple calls to `NCVPT` or `NCVPTC` instead.

To use `ncrecput` properly, you must know the number, order, and types of record variables in the netCDF file, information that can be determined with a call to `ncrecinq`. If your assumptions about the number, order, or types of record variables in the file is incorrect, calling this function may lead to incorrect results or a memory access error. **Warning: neither the compiler nor the netCDF software can detect errors with the pointer array argument to `ncrecput`.**

In case of a detected error, `ncrecput` returns -1. Possible causes of detectable errors include:

- The specified record number is less than zero.
- The specified netCDF file is in define mode rather than data mode.
- The specified netCDF ID does not refer to an open netCDF file.

C Interface: ncrecput

```
int ncrecput(int ncid, long recnum, const void *datap[]);
```

`ncid` NetCDF ID, returned from a previous call to `ncopen` or `nccreate`.

`recnum` Record number, specifying the value of the unlimited dimension for which data is to be written. The first record is record number `0`. Note that if you specify a value for `recnum` that is larger than the current size of the unlimited dimension, intervening records will be written with fill values before the data is written in the specified record, unless `ncsetfill` has been called to specify no prefilling.

`datap` Array of pointers to blocks of data values to be written, one for each record variable. `datap[i]`, if non-null, must point to an entire record's worth of data for the `i`-th record variable. For null pointers, no data will be written for the corresponding record variables. This permits you to specify an arbitrary subset of record variables. The data pointed to should be of the appropriate type for each record variable. **Warning: neither the compiler nor the netCDF software can detect whether the wrong type of data is used.**

Here is an example using `ncrecput` to write the value of a C struct into a netCDF file with a single call. This example assumes that record variables of the appropriate shapes and types have previously been created in the netCDF file.

```
#include <netcdf.h>
   ...
    static struct {
    char city[20];
    nclong date;
    float lat;
    float lon;
    float precip[24];           /* hourly precipitation */
    } rec = {
        "Pocatello",
        930228,
        42.92,
        -112.60,
        {0,0,.1,.2,.2,.3,.2,0,0,0,0,0,0,0,0,.3,1.1,0,0,0,0,0,0}
    };

    int ncid;                   /* id of open netcdf file */
    long recnum;                /* number of record to write */
    void *datap[5];             /* array of address pointers for record
vars */
    ...
    datap[0] = &rec.city[0];
    datap[1] = &rec.date;
    datap[2] = &rec.lat;
    datap[3] = &rec.lon;
    datap[4] = &rec.precip[0];

    ncrecput(ncid, recnum, datap);  /* instead of 5 calls to ncvarput */
```

7.9 Read a Single Data Value: ncvarget1, NCVGT1, and NCVG1C

The function `ncvarget1` (or `NCVGT1` or `NCVG1C` for FORTRAN) gets a single data value from a variable of an open netCDF file that is in data mode. Inputs are the netCDF ID, the variable ID, a multidimensional index that specifies which value to get, and the address of a location into which the data value will be read.

In case of an error, `ncvarget1` returns -1; `NCVGT1` returns a nonzero value in `rcode`. Possible causes of errors include:

- The variable ID is invalid for the specified netCDF file.
- The specified indices were out of range for the rank of the specified variable. For example, a negative index or an index that is larger than the corresponding dimension size will cause an error.
- The specified netCDF is in define mode rather than data mode.
- The specified netCDF ID does not refer to an open netCDF file.

C Interface: ncvarget1

```
int ncvarget1(int ncid, int varid, const long mindex[], void *value);
```

`ncid` NetCDF ID, returned from a previous call to `ncopen` or `nccreate`.

`varid` Variable ID.

`mindex` The multidimensional index of the the data value to be read. The indices are relative to 0, so for example, the first data value of a two-dimensional variable would have index `(0,0)`. The elements of `mindex` must correspond to the variable's dimensions. Hence, if the variable is a record variable, the first index is the record number.

`value` Pointer to the location into which the data value is read. The pointer is declared to be of the type `void *` because it can point to data of any of the basic netCDF types. The data should be of the appropriate type for the netCDF variable. **Warning: neither the compiler nor the netCDF software can detect whether the wrong type for the data value is used.**

Here is an example using `ncvarget1` to get the `(1,2,3)` element of the variable named `rh` in an existing netCDF file named '`foo.nc`'. For simplicity in this example, we assume that we know that `rh` is dimensioned with `time`, `lat`, and `lon`, so we want to get the value of `rh` that corresponds to the second `time` value, the third `lat` value, and the fourth `lon` value:

```
#include <netcdf.h>
   ...
int ncid;                               /* netCDF ID */
int rh_id;                              /* variable ID */
static long rh_index[] = {1, 2, 3};  /* where to get value from */
double rh_val;                          /* where to put it */
   ...
ncid = ncopen("foo.nc", NC_NOWRITE);
   ...
rh_id = ncvarid (ncid, "rh");
   ...
ncvarget1(ncid, rh_id, rh_index, &rh_val);
```

FORTRAN Interface: NCVGT1

```
      SUBROUTINE NCVGT1 (INTEGER NCID, INTEGER VARID,
     + INTEGER MINDEX(*), type VALUE,
     + INTEGER RCODE)

      SUBROUTINE NCVG1C (INTEGER NCID, INTEGER VARID,
     + INTEGER MINDEX(*), CHARACTER CHVAL,
     + INTEGER RCODE)
```

There are two FORTRAN subroutines, `NCVGT1` and `NCVG1C`, for reading a single value from a variable. The first reads a numeric value in a variable of numeric type, and the second reads a character value in a variable of character type.

`NCID` NetCDF ID, returned from a previous call to `NCOPN` or `NCCRE`.

`VARID` Variable ID.

`MINDEX` The multidimensional index of the the data value to be read. The indices are relative to 1, so for example, the first data value of a two-dimensional variable has index `(1,1)`. The elements of `mindex` correspond to the variable's dimensions. Hence, if the variable is a record variable, the last index is the record number.

`VALUE` For `NCVGT1`, the location into which the data value will be read. The data may be of a type corresponding to any of the netCDF types `NCSHORT`, `NCLONG`, `NCFLOAT`, or `NCDOUBLE`, but must be appropriate for the type of the netCDF variable. **Warning: neither the compiler nor the netCDF software can detect whether the wrong type of data is used.**

`CHVAL` For `NCVG1C`, the location into which the data value will be read. This should be of a type character, corresponding to the netCDF types `NCCHAR` or `NCBYTE`.

`RCODE` Returned error code. If no errors occurred, 0 is returned.

Here is an example using `NCVGT1` to get the `(4,3,2)` element of the variable named `rh` in an existing netCDF file named '`foo.nc`'. For simplicity in this example, we assume that we know that `rh` is dimensioned with `lon`, `lat`, and `time`, so we want to get the value of `rh` that corresponds to the fourth `lon` value, the third `lat` value, and the second `time` value:

```
INCLUDE 'netcdf.inc'
   ...
INTEGER NCID, RCODE
INTEGER RHID ! variable ID
INTEGER RHINDX(3) ! where to get value
DOUBLE PRECISION RHVAL ! put it here
DATA RHINDX /4, 3, 2/
   ...
NCID = NCOPN ('foo.nc', NCNOWRIT, RCODE)
   ...
RHID = NCVID (NCID, 'rh', RCODE)   ! get ID
CALL NCVGT1 (NCID, RHID, RHINDX, RHVAL, RCODE)
```

7.10 Read an Array of Values: ncvarget and NCVGT(C)

The function `ncvarget` (or `NCVGT` or `NCVGTC` for FORTRAN) reads an array of values from a netCDF variable of an open netCDF file. The array is specified by giving a corner and a vector of edge lengths. The values are read into consecutive locations with the last (or first for FORTRAN) dimension varying fastest. The netCDF file must be in data mode. [2]

In case of an error, `ncvarget` returns -1; `NCVGT` returns a nonzero value in `rcode`. Possible causes of errors include:

- The variable ID is invalid for the specified netCDF file.
- The specified corner indices were out of range for the rank of the specified variable. For example, a negative index or an index that is larger than the corresponding dimension size will cause an error.
- The specified edge lengths added to the specified corner would have referenced data out of range for the rank of the specified variable. For example, an edge length that is larger than the corresponding dimension size minus the corner index will cause an error.
- The specified netCDF is in define mode rather than data mode.
- The specified netCDF ID does not refer to an open netCDF file.

C Interface: ncvarget

```
int ncvarget(int ncid, int varid, const long start[], const long count[],
             void *values);
```

`ncid` NetCDF ID, returned from a previous call to `ncopen` or `nccreate`.

`varid` Variable ID.

`start` A vector of long integers specifying the index in the variable where the first of the data values will be read. The indices are relative to 0, so for example, the first data value of a variable would have index `(0, 0, ..., 0)`. The size of `start` must be the same as the number of dimensions of the specified variable. The elements of `start` correspond, in order, to the variable's dimensions. Hence, if the variable is a record variable, the first index would correspond to the starting record number for reading the data values.

`count` A vector of long integers specifying the edge lengths along each dimension of the block of data values to be read. To read a single value, for example, specify `count` as `(1, 1, ..., 1)`. The size of `count` is the number of dimensions of the specified variable. The elements of `count` correspond, in order, to the variable's dimensions. Hence, if the variable is a record variable, the first element of `count` corresponds to a count of the number of records to read.

`value` Pointer to the first of the locations into which the data values will be read. The order in which the data will be read from the netCDF variable is with the last dimension varying fastest. The pointer is declared to be of the type `void *` because it can point

[2] The current implementation of XDR on MSDOS systems restricts the amount of data accessed to no more than 64 Kbytes for each call to `ncvarget` (or `NCVGT` or `NCVGTC` for FORTRAN).

to data of any of the basic netCDF types. The data should be of the appropriate type for the netCDF variable. **Warning: neither the compiler nor the netCDF software can detect whether the wrong type of data is used.**

Here is an example using `ncvarget` to read all the values of the variable named `rh` from an existing netCDF file named '`foo.nc`'. For simplicity in this example, we assume that we know that `rh` is dimensioned with `time`, `lat`, and `lon`, and that there are three `time` values, five `lat` values, and ten `lon` values.

```
#include <netcdf.h>
   ...
#define TIMES 3
#define LATS 5
#define LONS 10
int ncid; /* netCDF ID */
int rh_id; /* variable ID */
static long start[] = {0, 0, 0}; /* start at first value */
static long count[] = {TIMES, LATS, LONS};
double rh_vals[TIMES*LATS*LONS]; /* array to hold values */
   ...
ncid = ncopen("foo.nc", NC_NOWRITE);
   ...
rh_id = ncvarid (ncid, "rh");
   ...
/* read values from netCDF variable */
ncvarget(ncid, rh_id, start, count, rh_vals);
```

FORTRAN Interface: NCVGT, NCVGTC

```
 SUBROUTINE NCVGT (INTEGER NCID, INTEGER VARID,
+ INTEGER START(*), INTEGER COUNT(*),
+ type VALUES, INTEGER RCODE)

 SUBROUTINE NCVGTC(INTEGER NCID, INTEGER VARID,
+ INTEGER START(*), INTEGER COUNTS(*),
+ CHARACTER*(*) STRING, INTEGER LENSTR,
+ INTEGER RCODE)
```

There are two FORTRAN subroutines, `NCVGT` and `NCVGTC`, for reading an array of values from a netCDF variable. The first reads numeric values from a variable of numeric type, and the second reads character values from a variable of character type.

`NCID`	NetCDF ID, returned from a previous call to `NCOPN` or `NCCRE`.
`VARID`	Variable ID.
`START`	A vector of integers specifying the index in the variable where the first of the data values will be read. The indices are relative to 1, so for example, the first data value of a variable would have index `(1, 1, ..., 1)`. The size of `START` must be the same as

the number of dimensions of the specified variable. The elements of `START` correspond, in order, to the variable's dimensions. Hence, if the variable is a record variable, the last index would correspond to the starting record number for reading the data values.

`COUNT` A vector of integers specifying the edge lengths along each dimension of the block of data values to be read. To read a single value, for example, specify `COUNT` as `(1, 1, ..., 1)`. The size of `COUNT` is the number of dimensions of the specified variable. The elements of `COUNT` correspond, in order, to the variable's dimensions. Hence, if the variable is a record variable, the last element of `COUNT` corresponds to a count of the number of records to read.

`VALUES` For `NCVGT`, the locations into which the data values will be read. The order in which the data will be read from the netCDF variable is with the first dimension varying fastest (like the ordinary FORTRAN convention). The data may be of a type corresponding to any of the netCDF types `NCSHORT`, `NCLONG`, `NCFLOAT`, or `NCDOUBLE`, but must be appropriate for the type of the netCDF variable. **Warning: neither the compiler nor the netCDF software can detect whether the wrong type of data is used.**

`STRING` For `NCVGTC`, the character string into which the character data will be read. The order in which the characters will be read from the netCDF variable is with the first dimension varying fastest (like the FORTRAN convention). The data may be of a type corresponding to the netCDF types `NCCHAR` or `NCBYTE`.

`LENSTR` For `NCVGTC`, the total declared length (in characters) of the `STRING` argument. This should be at least as large as the product of the elements of the `COUNT` vector. Note that this is not necessarily the same as the value returned by the FORTRAN `LEN` function, because an array argument may be provided. `NCVGTC` will check to make sure the requested data will fit in `LENSTR` characters.

`RCODE` Returned error code. If no errors occurred, 0 is returned.

Here is an example using `NCVGT` to read all the values of the variable named `rh` from an existing netCDF file named '`foo.nc`'. For simplicity in this example, we assume that we know that `rh` is dimensioned with `lon`, `lat`, and `time`, and that there are ten `lon` values, five `lat` values, and three `time` values.

```
INCLUDE 'netcdf.inc'
   ...
PARAMETER (NDIMS=3) ! number of dimensions
PARAMETER (TIMES=3, LATS=5, LONS=10) ! dimension sizes
INTEGER NCID, RCODE
INTEGER RHID ! variable ID
INTEGER START(NDIMS), COUNT(NDIMS)
DOUBLE RHVALS(LONS, LATS, TIMES)
DATA START /1, 1, 1/ ! start at first value
DATA COUNT /LONS, LATS, TIMES/
   ...
NCID = NCOPN ('foo.nc', NCNOWRIT, RCODE)
   ...
RHID = NCVID (NCID, 'rh', RCODE)! get ID
CALL NCVGT (NCID, RHID, START, COUNT, RHVALS, RCODE)
```

7.11 Read a Subsampled Or Mapped Array of Values: ncvargetg, NCVGTG and NCVGGC

The function `ncvargetg` (or `NCVGTG` or `NCVGGC` for FORTRAN) reads a subsampled or mapped array section of values from a netCDF variable of an open netCDF file. The subsampled or mapped array section is specified by giving a corner, a vector of edge lengths, a stride vector, and an index mapping vector. The values are read with the last (or first for FORTRAN) dimension of the netCDF variable varying fastest. The netCDF file must be in data mode.

In case of an error, `ncvargetg` returns -1; `NCVGTG` and `NCVGGC` return a nonzero value in `rcode`. Possible causes of errors include:

- The variable ID is invalid for the specified netCDF file.
- The specified corner indices were out of range for the rank of the specified variable. For example, a negative index or an index that is larger than the corresponding dimension size will cause an error.
- The specified edge lengths and strides added to the specified corner would have referenced data out of range for the rank of the specified variable. For example, an edge length that is larger than the corresponding dimension size minus the corner index will cause an error, as will accessing two or more points using a stride that is greater than the size of the netCDF variable in the corresponding dimension.
- A non-positive stride.
- The specified netCDF is in define mode rather than data mode.
- The specified netCDF ID does not refer to an open netCDF file.

C Interface: ncvargetg

```
int ncvargetg(int ncid, int varid, const long start[], const long count[],
                    const long stride[], const long imap[], void *values);
```

`ncid` NetCDF ID, returned from a previous call to `ncopen` or `nccreate`.

`varid` Variable ID.

`start` A vector of long integers specifying the index in the variable where the first of the data values will be read. The indices are relative to 0, so for example, the first data value of a variable would have index `(0, 0, ..., 0)`. The elements of `start` correspond, in order, to the variable's dimensions. Hence, if the variable is a record variable, the first index corresponds to the starting record number for reading the data values.

`count` A vector of long integers specifying the edge lengths along each dimension of the block of the block of data values to be read. To read a single value, for example, specify `count` as `(1, 1, ..., 1)`. The elements of `count` correspond, in order, to the variable's dimensions. Hence, if the variable is a record variable, the first element of `count` corresponds to a count of the number of records to read.

`stride` A vector of long integers specifying, for each dimension, the interval between the accessed values of a netCDF variable. The elements of the stride vector correspond, in order, to the variable's dimensions. A value of 1 accesses adjacent values of the netCDF variable in the corresponding dimension; a value of 2 accesses every other value of the netCDF variable in the corresponding dimension; and so on. A `NULL` stride argument obtains the default behavior in which adjacent values are accessed along each dimension.

`imap` A vector of long integers specifying, for each dimension, how data values associated with a netCDF variable are arranged in memory. The offset, in bytes, from the memory location pointed to by the `value` argument to a particular datum is given by the inner product of the index mapping vector with the coordinates of the datum. (The *inner product* of two vectors [x0, x1, ..., xn] and [y0, y1, ..., yn] is just x0*y0 + x1*y1 + ... + xn*yn.) The vector may contain negative values if the `value` argument is appropriately specified. A `NULL` argument obtains the default behavior in which the memory-resident values are assumed to have the same structure as the associated netCDF variable.

`value` Pointer to the first of the locations into which the data values will be read. The order in which the data will be read from the netCDF variable is with the last dimension varying fastest. The pointer is declared to be of the type `void *` because it can point to data of any of the basic netCDF types. The data should be of the appropriate type for the netCDF variable. **Warning: neither the compiler nor the netCDF software can detect whether the wrong type of data is used.**

Here is an example using `ncvargetg` to read every other value in each dimension of the variable named `rh` from an existing netCDF file named '`foo.nc`'. Values are assigned, using the same dimensional strides, to points in a 3-dimensional array of structures whose dimensions are the reverse of the netCDF variable. For simplicity in this example, we assume that we know that `rh` is dimensioned with `time`, `lat`, and `lon`, and that there are three `time` values, five `lat` values, and ten `lon` values.

```
#include <netcdf.h>
   ...
#define TIMES 3
#define LATS  5
#define LONS 10
int ncid;                           /* netCDF ID */
int rh_id;                          /* variable ID */
static long start[] = {0, 0, 0};  /* start at first value */
static long count[] = {TIMES, LATS, LONS};
static long stride[] = {2, 2, 2}; /* every other value */
long imap[3];                       /* set to reverse of variable */
struct datum {
    int dummy;                      /* to illustrate mapping vector usage */
    double rh_val;                  /* actual value to be read */
}      data[TIMES][LATS][LONS];    /* array to hold values */
   ...
   ncid = ncopen("foo.nc", NC_NOWRITE);
   ...
   rh_id = ncvarid (ncid, "rh");
   ...
   /* access every 'stride' in-memory value using reversed dimensions */
   imap[0] = stride[2]*sizeof(struct datum);
   imap[1] = stride[1]*(1+(LONS-1)/stride[0])*imap[0];
   imap[2] = stride[0]*(1+(LATS-1)/stride[1])*imap[1];
   /* read values from netCDF variable */
   ncvargetg(ncid, rh_id, start, count, stride, imap, &data[0][0][0].rh_val);
   ...
```

FORTRAN Interface: NCVGTG, NCVGGC

```
      SUBROUTINE NCVGTG (INTEGER NCID, INTEGER VARID,
     + INTEGER START(*), INTEGER COUNT(*),
     + INTEGER STRIDE(*), INTEGER IMAP(*),
     + type VALUES, INTEGER RCODE)

      SUBROUTINE NCVGGC (INTEGER NCID, INTEGER VARID,
     + INTEGER START(*), INTEGER COUNT(*),
     + INTEGER STRIDE(*), INTEGER IMAP(*),
     + CHARACTER*(*) STRING, INTEGER RCODE)
```

There are two FORTRAN subroutines, `NCVGTG` and `NCVGGC`, for reading a subsampled or mapped array section of values from a netCDF variable. The first reads numeric values from a variable of numeric type, and the second reads character values from a variable of character type.

`NCID` NetCDF ID, returned from a previous call to `NCOPN` or `NCCRE`.

`VARID`	Variable ID.
`START`	A vector of integers specifying the index in the variable from which the first of the data values will be read. The indices are relative to 1, so for example, the first data value of a variable would have index `(1, 1, ..., 1)`. The elements of `START` correspond, in order, to the variable's dimensions. Hence, if the variable is a record variable, the last index would correspond to the starting record number for reading the data values.
`COUNT`	A vector of integers specifying the edge lengths along each dimension of the block of data values to be read. To read a single value, for example, specify `COUNT` as `(1, 1, ..., 1)`. The elements of `COUNT` correspond, in order, to the variable's dimensions. Hence, if the variable is a record variable, the last element of `COUNT` corresponds to a count of the number of records to read.
`STRIDE`	A vector of integers specifying, for each dimension, the interval between the accessed values of a netCDF variable or the value `0`. The elements of the vector correspond, in order, to the variable's dimensions. A value of 1 accesses adjacent values of the netCDF variable in the corresponding dimension; a value of 2 accesses every other value of the netCDF variable in the corresponding dimension; and so on. An `0` argument obtains the default behavior in which adjacent values are accessed along each dimension.
`IMAP`	A vector of long integers specifying, for each dimension, how data values associated with a netCDF variable are arranged in memory or the value `0`. The offset, in bytes, from the memory location pointed to by the `value` argument to a particular datum is given by the inner product of the index mapping vector with the (origin-0) coordinates of the datum. (The *inner product* of two vectors [x1, x2, ..., xn] and [y1, y2, ..., yn] is just x1*y1 + x2*y2 + ... + xn*yn.) The vector may contain negative values if the `value` argument is appropriately specified. A `0` argument obtains the default behavior in which the memory-resident values are assumed to have the same structure as the associated netCDF variable.
`VALUES`	For `NCVGTG`, the locations into which the data values will be read. The order in which the data will be read from the netCDF variable is with the first dimension varying fastest (like the ordinary FORTRAN convention). The data may be of a type corresponding to any of the netCDF types `NCSHORT`, `NCLONG`, `NCFLOAT`, or `NCDOUBLE`, but must be appropriate for the type of the netCDF variable. **Warning: neither the compiler nor the netCDF software can detect whether the wrong type of data is used.**
`STRING`	For `NCVGGC`, the character string into which the character data will be read. The order in which the characters will be read from the netCDF variable is with the first dimension varying fastest (like the FORTRAN convention). The data may be of a type corresponding to the netCDF types `NCCHAR` or `NCBYTE`.
`RCODE`	Returned error code. If no errors occurred, 0 is returned.

Here is an example using `NCVGTG` to read every other value in each dimension of the variable named `rh` from an existing netCDF file named '`foo.nc`'. Values are assigned, using the same dimensional strides, to a 2-parameter array whose dimensions are the reverse of the netCDF variable. For simplicity in this example, we assume that we know that `rh` is dimensioned with `lon`, `lat`, and `time`, and that there are ten `lon` values, five `lat` values, and three `time` values.

```
 INCLUDE 'netcdf.inc'
    ...
 PARAMETER (NDIMS=3) ! number of dimensions
 PARAMETER (TIMES=3, LATS=5, LONS=10) ! dimension sizes
 INTEGER NCID, RCODE
 INTEGER RHID ! variable ID
 INTEGER START(NDIMS), COUNT(NDIMS),
+ STRIDE(NDIMS), IMAP(NDIMS)
 DOUBLE DATA(2, TIMES, LATS, LONS) ! rh is second parameter
 DATA START /1, 1, 1/ ! start at first value
 DATA COUNT /LONS, LATS, TIMES/
 DATA STRIDE /2, 2, 2/
    ...
 NCID = NCOPN ('foo.nc', NCNOWRIT, RCODE)
    ...
 RHID = NCVID (NCID, 'rh', RCODE)  ! get ID
 IMAP(3) = 8*2*2 ! every other point of vector of 2-doubles
 IMAP(2) = IMAP(3)*(1+(TIMES-1)/STRIDE(3))*2
 IMAP(1) = IMAP(2)*(1+(LATS-1)/STRIDE(2))*2
 CALL NCVGTG (NCID, RHID, START, COUNT, STRIDE, IMAP,
+ DATA(2,1,1,1), RCODE)
```

7.12 Get a Record: ncrecget

The function `ncrecget` reads a multi-variable record of values (or part of a record of values) from the record variables of an open netCDF file. The record is specified by giving a record number. The locations into which the data will be read are specified by an array of pointers, one for each record variable, to blocks of data. Each block of data should be of the appropriate size and type for a record's worth of data for the corresponding record variable. Each such pointer must be either '`0`' (a null pointer), to indicate that no data is to be read for that variable, or must point to space for an entire record's worth of data of the appropriate type for the corresponding record variable. The values for each record variable will be ordered with the last dimension of the netCDF variable varying fastest. The netCDF file must be in data mode.

The `ncrecget` function is not strictly necessary, since the same data may be read with a sequence of calls to `ncvarget`, one for each record variable for which a non-null pointer is specified. This function is provided in the C interface for convenience only; no corresponding FORTRAN interface is available, so FORTRAN users should use multiple calls to `NCVGT` or `NCVGTC` instead.

To use `ncrecget` properly, you must know the number, order, and types of record variables in the netCDF file, information that can be determined with a call to `ncrecinq`. If your assumptions about the number, order, or types of record variables in the file is incorrect, calling this function may lead to incorrect results or a memory access error. **Warning: neither the compiler nor the netCDF software can detect errors with the pointer array argument to `ncrecget`.**

In case of a detected error, `ncrecget` returns -1. Possible causes of detectable errors include:

- The specified record number is less than zero.
- The specified netCDF file is in define mode rather than data mode.
- The specified netCDF ID does not refer to an open netCDF file.

C Interface: ncrecget

```
int ncrecget(int ncid, long recnum, void *datap[]);
```

`ncid` NetCDF ID, returned from a previous call to `ncopen` or `nccreate`.

`recnum` Record number, specifying the value of the unlimited dimension for which data is to be read. The first record is record number `0`.

`datap` Array of pointers to blocks of data into which the requested values will be read, one for each record variable. `datap[i]`, if non-null, must point to enough space to hold an entire record's worth of data for the `i`-th record variable. For null pointers, no data will be read for the corresponding record variables. This permits you to specify an arbitrary subset of record variables. The data pointed to should be of the appropriate type for each record variable. **Warning: neither the compiler nor the netCDF software can detect whether the wrong type of data is used.**

Here is an example using `ncrecget` to read values into several C arrays and scalars with a single call. This example assumes that record variables of the appropriate shapes and types have previously been created in the netCDF file.

```
#include <netcdf.h>
   ...
    static struct {
    char city[20];
    nclong date;
    float lat;
    float lon;
    float precip[24];
    } rec[10];

    int ncid;           /* id of open netcdf file */
    long i;             /* number of record to read */
    void *datap[5];     /* array of address pointers for record vars */
    ...

    /* Get first 10 records of data */
    for(i=0; i<10; i++) {
        datap[0] = &rec[i].city[0];
        datap[1] = &rec[i].date;
        datap[2] = &rec[i].lat;
        datap[3] = &rec[i].lon;
        datap[4] = &rec[i].precip[0];
        ncrecget(ncid, i, datap);  /* instead of 5 calls to ncvarget */
    }
```

7.13 Reading and Writing Character String Values

Character strings are not a primitive netCDF data type, in part because FORTRAN does not support the abstraction of variable-length character strings (the FORTRAN `LEN` function returns the static length of a character string, not its dynamic length). As a result, a character string cannot be written or read as a single object in the netCDF interface. Instead, a character string must be treated as an array of characters, and array access must be used to read and write character strings as variable data in netCDF files. Furthermore, variable-length strings are not supported by the netCDF interface except by convention; for example, you may treat a zero byte as terminating a character string, but you must explicitly specify the length of strings to be read from and written to netCDF variables.

Character strings as attribute values are easier to use, since the strings are treated as a single unit for access. However, the value of a character-string attribute is still an array of characters with an explicit length that must be specified when the attribute is defined.

When you define a variable that will have character-string values, use a *character-position dimension* as the most quickly varying dimension for the variable (the last dimension for the variable in C, the first in FORTRAN). The size of the character-position dimension will be the maximum string length of any value to be stored in the character-string variable. Space for maximum-size strings will be allocated in the disk representation of character-string variables whether you use the space or not. If two or more variables have the same maximum length, the same character-position dimension may be used in defining the variable shapes.

To write a character-string value into a character-string variable, use array access. This requires that you specify both a corner and a vector of edge lengths. The character-position dimension at the corner should be zero (one for FORTRAN). If the length of the string to be written is `n`, then the vector of edge lengths will specify `n` in the character-position dimension, and one for all the other dimensions, i.e., `(1, 1, ..., 1, n)` or `(n, 1, 1, ..., 1)` in FORTRAN.

C Interface

In C, fixed-size strings may be written to a netCDF file without the terminating zero byte, to save space. Variable-length strings should be written *with* a terminating zero byte so that the intended length of the string can be determined when it is later read.

Here is an example that defines a record variable, `tx`, for character strings and stores a character-string value into the third record using `ncvarput`. In this example, we assume the string variable and data are to be added to an existing netCDF file named '`foo.nc`' that already has an unlimited record dimension `time`.

```
#include <netcdf.h>
   ...
int  ncid;            /* netCDF ID */
int  chid;            /* dimension ID for char positions */
int  timeid;          /* dimension ID for record dimension */
int  tx_id;           /* variable ID */
#define TDIMS 2       /* rank of tx variable */
int tx_dims[TDIMS];   /* variable shape */
long tx_start[TDIMS];
long tx_count[TDIMS];
static char tx_val[] =
        "example string"; /* string to be put */
   ...
ncid = ncopen("foo.nc", NC_WRITE);
ncredef(ncid);       /* enter define mode */
   ...
/* define character-position dimension for strings of max length 40 */
chid = ncdimdef(ncid, "chid", 40L);
   ...
/* define a character-string variable */
tx_dims[0] = timeid;
tx_dims[1] = chid;    /* character-position dimension last */
tx_id = ncvardef (ncid, "tx", NC_CHAR, TDIMS, tx_dims);
   ...
ncendef(ncid);       /* leave define mode */
   ...
/* write tx_val into tx netCDF variable in record 3 */
```

```
tx_start[0] = 3;      /* record number to write */
tx_start[1] = 0;      /* start at beginning of variable */
tx_count[0] = 1;      /* only write one record */
tx_count[1] = strlen(tx_val) + 1;  /* number of chars to write */
ncvarput(ncid, tx_id, tx_start, tx_count, tx_val);
```

FORTRAN Interface

In FORTRAN, fixed-size strings may be written to a netCDF file without a terminating character, to save space. Variable-length strings should follow the C convention of writing strings with a terminating zero byte so that the intended length of the string can be determined when it is later read by either C or FORTRAN programs.

The FORTRAN interface for reading and writing strings requires the use of different subroutines for accessing string values and numeric values, because standard FORTRAN does not permit the same formal parameter to be used for both character values and numeric values. An additional argument, specifying the declared length of the character string passed as a value, is required for `NCVPTC` and `NCVGTC`. The actual length of the string is specified as the value of the edge-length vector corresponding to the character-position dimension.

Here is an example that defines a record variable, `tx`, for character strings and stores a character-string value into the third record using `NCVPTC`. In this example, we assume the string variable and data are to be added to an existing netCDF file named '`foo.nc`' that already has an unlimited record dimension `time`.

```
      INCLUDE 'netcdf.inc'
         ...
      INTEGER   TDIMS, TXLEN
      PARAMETER (TDIMS=2)    ! number of TX dimensions
      PARAMETER (TXLEN = 15) ! length of example string
      INTEGER  NCID, RCODE
      INTEGER  CHID          ! char position dimension id
      INTEGER  TIMEID        ! record dimension id
      INTEGER  TXID          ! variable ID
      INTEGER  TXDIMS(TDIMS) ! variable shape
      INTEGER  TSTART(TDIMS), TCOUNT(TDIMS)
      CHARACTER*40 TXVAL     ! max length 40
      DATA TXVAL /'example string'/
         ...
      TXVAL(TXLEN:TXLEN) = CHAR(0)   ! null terminate
         ...
      NCID = NCOPN('foo.nc', NCWRITE, RCODE)
      CALL NCREDF(NCID, RCODE) ! enter define mode
         ...
* define character-position dimension for strings of max length 40
      CHID = NCDDEF(NCID, "chid", 40, RCODE)
         ...
```

```
* define a character-string variable
      TXDIMS(1) = CHID   ! character-position dimension first
      TXDIMS(2) = TIMEID
      TXID = NCVDEF(NCID, "tx", NCCHAR, TDIMS, TXDIMS, RCODE)
         ...
      CALL NCENDF(NCID, RCODE) ! leave define mode
         ...
* write txval into tx netCDF variable in record 3
      TSTART(1) = 0      ! start at beginning of variable
      TSTART(2) = 3      ! record number to write
      TCOUNT(1) = TXLEN  ! number of chars to write
      TCOUNT(2) = 1      ! only write one record
      CALL NCVPTC (NCID, TXID, TSTART, TCOUNT, TXVAL, 40, RCODE)
```

7.14 Fill Values

What happens when you try to read a value that was never written in an open netCDF file? You might expect that this should always be an error, and that you should get an error message or an error status returned. You *do* get an error if you try to read data from a netCDF file that is not open for reading, if the variable ID is invalid for the specified netCDF file, or if the specified indices are not properly within the range defined by the dimension sizes of the specified variable. Otherwise, reading a value that was not written returns a special *fill value* used to fill in any undefined values when a netCDF variable is first written.

You may ignore fill values and use the entire range of a netCDF data type, but in this case you should make sure you write all data values before reading them. If you know you will be writing all the data before reading it, you can specify that no prefilling of variables with fill values will occur by calling `ncsetfill` before writing. This may provide a significant performance gain for netCDF writes.

The variable attribute `_FillValue` is used to specify the fill value. The default fill values for each type are defined in the include file '`netcdf.h`' (or '`netcdf.inc`' for FORTRAN).

One difference between the netCDF byte and character types is that the two types have different default fill values. The default fill value for characters is the zero byte, a useful value for detecting the end of variable-length C character strings. If you need a fill value for a byte variable, it is recommended that you explicitly define an appropriate `_FillValue` attribute, as generic utilities such as `ncdump` will not assume a default fill value for byte variables.

7.15 Rename a Variable: ncvarrename and NCVREN

The function `ncvarrename` (or `NCVREN` for FORTRAN) changes the name of a netCDF variable in an open netCDF. If the new name is longer than the old name, the netCDF file must be in define mode. You cannot rename a variable to have the name of any existing variable.

In case of an error, `ncvarrename` returns -1; `NCVREN` returns a nonzero value in `rcode`. Possible causes of errors include:

- The new name is in use as the name of another variable.
- The variable ID is invalid for the specified netCDF file.
- The specified netCDF ID does not refer to an open netCDF file.

C Interface: ncvarrename

```
int ncvarrename(int ncid, int varid, const char* name);
```

`ncid`	NetCDF ID, returned from a previous call to `ncopen` or `nccreate`.
`varid`	Variable ID.
`name`	New name for the specified variable.

Here is an example using `ncvarrename` to rename the variable `rh` to `rel_hum` in an existing netCDF file named '`foo.nc`':

```
#include <netcdf.h>
   ...
int  ncid;                        /* netCDF ID */
int  rh_id;                       /* variable ID */
   ...
ncid = ncopen("foo.nc", NC_WRITE);
   ...
ncredef(ncid);  /* put in define mode to rename variable */
rh_id = ncvarid (ncid, "rh");
ncvarrename (ncid, rh_id, "rel_hum");
ncendef(ncid);  /* leave define mode */
```

FORTRAN Interface: NCVREN

```
      SUBROUTINE NCVREN (INTEGER NCID, INTEGER VARID,
     +                   CHARACTER*(*) NEWNAM, INTEGER RCODE)
```

`NCID`	NetCDF ID, returned from a previous call to `NCOPN` or `NCCRE`.
`VARID`	Variable ID.
`NEWNAM`	New name for the specified variable.

Here is an example using `NCVREN` to rename the variable `rh` to `rel_hum` in an existing netCDF file named '`foo.nc`':

```
INCLUDE 'netcdf.inc'
   ...
INTEGER  NCID, RCODE
INTEGER  RHID              ! variable ID
   ...
NCID = NCOPN ('foo.nc', NCWRITE, RCODE)
   ...
CALL NCREDF (CDFFID, RCODE)  ! enter definition mode
RHID = NCVID (NCID, 'rh', RCODE)  ! get ID
CALL NCVREN (NCID, RHID, 'rel_hum', RCODE)
CALL NCENDF (CDFFID, RCODE)  ! leave definition mode
```

7.16 Get Number of Bytes for a Data Type: nctypelen and NCTLEN

The function `nctypelen` (or `NCTLEN` for FORTRAN) returns the number of bytes per netCDF data type.

In case of an error, `nctypelen` returns -1; `NCTLEN` returns a nonzero value in `rcode`. One possible cause of errors is:

- The specified data type is not a valid netCDF data type.

C Interface: nctypelen

```
int nctypelen (nc_type datatype);
```

`datatype` One of the set of predefined netCDF data types. The type of this parameter, `nc_type`, is defined in the netCDF header file. The valid netCDF data types are `NC_BYTE`, `NC_CHAR`, `NC_SHORT`, `NC_LONG`, `NC_FLOAT`, and `NC_DOUBLE`.

Here is an example using `nctypelen` to determine how many bytes are required to store a single value of the variable `rh` in an existing netCDF file named '`foo.nc`':

```
#include <netcdf.h>
   ...
int  ncid;        /* netCDF ID */
int  rh_id;       /* variable ID */
nc_type rh_type;           /* variable type */
int rh_ndims;              /* number of dims */
int rh_dims[MAX_VAR_DIMS]; /* variable shape */
int rh_natts;              /* number of attributes */
int rhbytes;      /* number of bytes per value for "rh" */
   ...
ncid = ncopen("foo.nc", NC_NOWRITE);
   ...
rh_id = ncvarid (ncid, "rh");
/* get type.  we don't need name, since we already know it */
ncvarinq (ncid, rh_id, 0, &rh_type, &rh_ndims, rh_dims,
          &rh_natts);
rhbytes = nctypelen (rh_type);
```

FORTRAN Interface: NCTLEN

```
INTEGER FUNCTION NCTLEN (INTEGER TYPE ,INTEGER RCODE)
```

TYPE One of the set of predefined netCDF data types. The valid netCDF data types are NCBYTE, NCCHAR, NCSHORT, NCLONG, NCFLOAT, and NCDOUBLE.

RCODE Returned error code. If no errors occurred, 0 is returned.

Here is an example using NCTLEN to determine how many bytes are required to store a single value of the variable rh in an existing netCDF file named 'foo.nc':

```
      INCLUDE 'netcdf.inc'
         ...
      INTEGER  NCID              ! netCDF ID
      INTEGER  RHID               ! variable ID
      CHARACTER*31 RHNAME         ! variable name
      INTEGER  RHTYPE             ! variable type
      INTEGER  RHN                ! number of dimensions
      INTEGER  RHDIMS(MAXVDIMS)   ! variable shape
      INTEGER  RHNATT             ! number of attributes
      INTEGER  RHBYTS             ! bytes per value
         ...
      NCID = NCOPN ('foo.nc', NCNOWRIT, RCODE)
         ...
      RHID = NCVID (NCID, 'rh', RCODE)
* get type of "rh"
      CALL NCVINQ (NCID, RHID, RHNAME, RHTYPE, RHN, RHDIMS, RHNATT,
     +             RCODE)
      RHBYTS = NCTLEN (RHTYPE)
```

7.17 Get Information About Record Variables: ncrecinq

The function `ncrecinq` returns information about the record variables (variables that use the unlimited dimension) in a netCDF file. The information returned is the number of record variables, their variable IDs, and the size (in bytes) for a record's worth of data for each record variable.

The `ncrecinq` function is not strictly necessary, since the information it returns can be computed from information returned by `ncinquire`, `ncdiminq`, and `ncvarinq` functions or their FORTRAN counterparts. This function is provided in the C interface for convenience only, to assist in using the C functions `ncrecput` and `ncrecget`.

In case of an error, `ncrecinq` returns -1. Possible causes of errors include:

- The specified netCDF ID does not refer to an open netCDF file.

C Interface: ncrecinq

```
int ncrecinq(int ncid, int* nrvars, int rvarids[], long rsizes[]);
```

`ncid` NetCDF ID, returned from a previous call to `ncopen` or `nccreate`.

`nrvars` Number of record variables.

`rvarids` Returned vector of `nrvars` variable IDs for the record variables in this netCDF file. The caller must allocate enough space for a vector of at least `nrvars` integers to be returned. The maximum possible number of variable IDs returned is given by the predefined constant `MAX_NC_VARS`. If this parameter is given as '`0`' (a null pointer), no vector will be returned so no space to hold the record variable IDs needs to be declared or allocated.

`rsizes` Returned vector of `nrvars` sizes for the record variables in this netCDF file. The size of a record variable is the number of bytes required to hold a record's worth of data, which is the product of the non-record dimensions and the size of data type, in bytes. The caller must allocate enough space for a vector of at least `nrvars` longs to be returned. The maximum possible number of variable IDs returned is given by the predefined constant `MAX_NC_VARS`. If this parameter is given as '`0`' (a null pointer), no vector will be returned so no space to hold the record variable sizes needs to be declared or allocated.

Here is an example using `ncrecinq` to find out about the record variables in an existing netCDF file named '`foo.nc`':

```
#include <netcdf.h>
   ...
int  ncid;                     /* netCDF ID */
int  nrvars;                   /* number of record variables */
int  rvarids[MAX_NC_VARS];     /* IDs of record variables */
long rvarsizes[MAX_NC_VARS];   /* record sizes of record variables */
   ...
ncid = ncopen ("foo.nc", NC_NOWRITE);
   ...
ncrecinq (ncid, &nrvars, rvarids, rvarsizes);
```

8 Attributes

Attributes may be associated with each netCDF variable to specify such properties as units, special values, maximum and minimum valid values, scaling factors, and offsets. Attributes for a netCDF file are defined when the file is first created, while the netCDF file is in define mode. Additional attributes may be added later by reentering define mode. A netCDF attribute has a netCDF variable to which it is assigned, a name, a type, a length, and a sequence of one or more values. An attribute is designated by its variable ID and name, except in one case (`ncattname` or `NCANAM` in FORTRAN), where attributes are designated by variable ID and number because their names are unknown.

The attributes associated with a variable are typically defined right after the variable is created, while still in define mode. The data type, length, and value of an attribute may be changed even when in data mode, as long as the changed attribute requires no more space than the attribute as originally defined.

It is also possible to have attributes which are not associated with any variable. These are called *global attributes* and are identified by using `NC_GLOBAL` as a variable pseudo-ID. Global attributes are usually related to the netCDF file as a whole and may be used for purposes such as providing a title or processing history for a netCDF dataset.

Operations supported on attributes are:

- Create an attribute, given its variable ID, name, data type, length, and value.
- Get attribute's data type and length from its variable ID and name.
- Get attribute's value from its variable ID and name.
- Copy attribute from one netCDF variable to another.
- Get name of attribute from its number.
- Rename an attribute.
- Delete an attribute.

8.1 Attribute Conventions

Names commencing with underscore ('`_`') are reserved for use by the netCDF library. Most generic applications that process netCDF files assume standard attribute conventions and it is strongly recommended that these be followed unless there are good reasons for not doing so. Below we list the names and meanings of recommended standard attributes that have proven useful. Note that some of these (e.g. `units`, `valid_range`, `scale_factor`) assume numeric data and should not be used with character data.

`units` A character string that specifies the units used for the variable's data. Unidata has developed a freely-available library of routines to convert between character string and binary forms of unit specifications and to perform various useful operations on the binary forms. This library is used in some netCDF applications. Using the recommended units syntax permits data represented in conformable units to be automatically converted to common units for arithmetic operations. See Appendix A [Units], page 141, for more information.

`long_name`
: A long descriptive name. This could be used for labeling plots, for example. If a variable has no `long_name` attribute assigned, the variable name should be used as a default.

`valid_min`
: A scalar specifying the minimum valid value for this variable.

`valid_max`
: A scalar specifying the maximum valid value for this variable.

`valid_range`
: A vector of two numbers specifying the minimum and maximum valid values for this variable, equivalent to specifying values for both `valid_min` and `valid_max` attributes. Any of these attributes define the *valid range*. The attribute `valid_range` must not be defined if either `valid_min` or `valid_max` is defined.

Generic applications should treat values outside the *valid range* as missing. The type of each `valid_range`, `valid_min` and `valid_max` attribute should match the type of its variable (except that for `byte` data, these can be of a signed integral type to specify the intended range).

If neither `valid_min`, `valid_max` nor `valid_range` is defined then generic applications should define a valid range as follows. If the data type is byte and `_FillValue` is not explicitly defined, then the valid range should include all possible values. Otherwise, the valid range should exclude the the `_FillValue` as follows. If the `_FillValue` is positive then it defines a valid maximum, otherwise it defines a valid minimum. For integer types, there should be a difference of 1 between the `_FillValue` and this valid minimum or maximum. For floating point types, the difference should be twice the minimum possible (1 in the least significant bit) to allow for rounding error.

`scale_factor`
: If present for a variable, the data are to be multiplied by this factor after the data are read by the application that accesses the data.

`add_offset`
: If present for a variable, this number is to be added to the data after it is read by the application that accesses the data. If both `scale_factor` and `add_offset` attributes are present, the data are first scaled before the offset is added. The attributes `scale_factor` and `add_offset` can be used together to provide simple data compression to

store low-resolution floating-point data as small integers in a netCDF file. When scaled data are written, the application should first subtract the offset and then divide by the scale factor.

When `scale_factor` and `add_offset` are used for packing, the associated variable (containing the packed data) is typically of type byte or short, whereas the unpacked values are intended to be of type float or double. The attributes `scale_factor` and `add_offset` should both be of the type intended for the unpacked data, e.g. float or double.

`_FillValue`
: The `_FillValue` attribute specifies the *fill value* used to pre-fill disk space allocated to the variable. Such pre-fill occurs unless *nofill mode* is set using `ncsetfill` (or `NCSFIL` for FORTRAN). See Section 5.10 [Set Fill Mode for Writes: ncsetfill and NCSFIL], page 48, for details. The *fill value* is returned when reading values that were never written. If `_FillValue` is defined then it should be scalar and of the same type as the variable. The `_FillValue` is typically outside the valid range and therefore treated by generic applications as missing. However it is legal for `_FillValue` to be within the valid range so that it is treated like any other valid value. It is not necessary to define your own `_FillValue` attribute for a variable if the default *fill value* for the type of the variable is adequate. However, use of the default fill value for data type byte is not recommended. Note that if you change the value of this attribute, the changed value applies only to subsequent writes; previously written data are not changed. See Section 7.14 [Fill Values], page 94, for more information.

`missing_value`
: This attribute is not treated in any special way by the library or conforming generic applications, but is often useful documentation and may be used by specific applications. The `missing_value` attribute can be a scalar or vector containing values indicating missing data. These values should all be outside the *valid range* so that generic applications will treat them as missing. See Section 7.14 [Fill Values], page 94, for more information.

`signedness`
: Deprecated attribute, originally designed to indicate whether byte values should be treated as signed or unsigned. The attributes `valid_min` and `valid_max` may be used for this purpose. For example, if you intend that a byte variable store only non-negative values, you can use `valid_min = 0` and `valid_max = 255`. This attribute is ignored by the netCDF library.

`C_format`
: A character array providing the format that should be used by C applications to print values for this variable. For example, if you know a variable is only accurate to three significant digits, it would be appropriate to define the `C_format` attribute as `"%.3g"`. The `ncdump` utility program uses this attribute for variables for which it is defined. The

format applies to the scaled (internal) type and value, regardless of the presence of the scaling attributes `scale_factor` and `add_offset`.

`FORTRAN_format`
: A character array providing the format that should be used by FORTRAN applications to print values for this variable. For example, if you know a variable is only accurate to three significant digits, it would be appropriate to define the `FORTRAN_format` attribute as `"(G10.3)"`.

`title`
: A global attribute that is a character array providing a succinct description of what is in the dataset.

`history`
: A global attribute for an audit trail. This is a character array with a line for each invocation of a program that has modified the file. Well-behaved generic netCDF applications should append a line containing: date, time of day, user name, program name and command arguments.

`Conventions`
: If present, 'Conventions' is a global attribute that is a character array for the name of the conventions followed by the file, in the form of a string that is interpreted as a directory name relative to a directory that is a repository of documents describing sets of discipline-specific conventions. This permits a hierarchical structure for conventions and provides a place where descriptions and examples of the conventions may be maintained by the defining institutions and groups. The conventions directory name is currently interpreted relative to the directory `pub/netcdf/Conventions/` on the host machine `ftp.unidata.ucar.edu`. Alternatively, a full URL specification may be used to name a WWW site where documents that describe the conventions are maintained.

For example, if a group named NUWG agrees upon a set of conventions for dimension names, variable names, required attributes, and netCDF representations for certain discipline-specific data structures, they may store a document describing the agreed-upon conventions in a file in the `NUWG/` subdirectory of the Conventions directory. Files that followed these conventions would contain a global `Conventions` attribute with value `"NUWG"`.

Later, if the group agrees upon some additional conventions for a specific subset of NUWG data, for example time series data, the description of the additional conventions might be stored in the `NUWG/Time_series/` subdirectory, and files that adhered to these additional conventions would use the global `Conventions` attribute with value `"NUWG/Time_series"`, implying that this file adheres to the NUWG conventions and also to the additional NUWG time-series conventions.

8.2 Create an Attribute: ncattput, NCAPT, and NCAPTC

The function `ncattput` (or `NCAPT` or `NCAPTC` for FORTRAN) adds or changes a variable attribute or global attribute of an open netCDF file. If this attribute is new, or if the space required to store the attribute is greater than before, the netCDF file must be in define mode.

In case of an error, `ncattput` returns -1; `NCAPT` returns a nonzero value in `rcode`. Possible causes of errors include:

- The variable ID is invalid for the specified netCDF file.
- The specified netCDF type is invalid.
- The specified length is negative.
- The specified open netCDF file is in data mode and the specified attribute would expand.
- The specified open netCDF file is in data mode and the specified attribute does not already exist.
- The specified netCDF ID does not refer to an open netCDF file.

C Interface: ncattput

```
int ncattput(int ncid, int varid, const char* name, nc_type datatype,
             int len, const void* values);
```

`ncid`	NetCDF ID, returned from a previous call to `ncopen` or `nccreate`.
`varid`	Variable ID of the variable to which the attribute will be assigned or `NC_GLOBAL` for a global attribute.
`name`	Attribute name. Must begin with an alphabetic character, followed by zero or more alphanumeric characters including the underscore ('`_`'). Case is significant. Attribute name conventions are assumed by some netCDF generic applications, e.g., `units` as the name for a string attribute that gives the units for a netCDF variable. See Section 8.1 [Attribute Conventions], page 101, for examples of attribute conventions.
`datatype`	One of the set of predefined netCDF data types. The type of this parameter, `nc_type`, is defined in the netCDF header file. The valid netCDF data types are `NC_BYTE`, `NC_CHAR`, `NC_SHORT`, `NC_LONG`, `NC_FLOAT`, and `NC_DOUBLE`.
`len`	Number of values provided for the attribute. If the attribute is of type `NC_CHAR`, this is one more than the string length (since the terminating zero byte is stored).
`values`	Pointer to one or more data values. The pointer is declared to be of the type `void *` because it can point to data of any of the basic netCDF types. The data should be of the appropriate type for the netCDF attribute. **Warning: neither the compiler nor the netCDF software can detect whether the wrong type of data is used.**

Here is an example using `ncattput` to add a variable attribute named `valid_range` for a netCDF variable named `rh` and a global attribute named `title` to an existing netCDF file named '`foo.nc`':

```
#include <netcdf.h>
   ...
int  ncid;                          /* netCDF ID */
int  rh_id;                         /* variable ID */
static double rh_range[] = {0.0, 100.0};  /* attribute vals */
static char title[] = "example netCDF file";
   ...
ncid = ncopen("foo.nc", NC_WRITE);
   ...
ncredef(ncid);                     /* enter define mode */
rh_id = ncvarid (ncid, "rh");
   ...
ncattput (ncid, rh_id, "valid_range", NC_DOUBLE, 2, rh_range);
ncattput (ncid, NC_GLOBAL, "title", NC_CHAR, strlen(title)+1,
          title);
   ...
ncendef(ncid);                     /* leave define mode */
```

FORTRAN Interface: NCAPT, NCAPTC

```
      SUBROUTINE NCAPT (INTEGER NCID, INTEGER VARID,
     +                  CHARACTER*(*) ATTNAM, INTEGER ATTYPE,
     +                  INTEGER ATTLEN, type VALUE,
     +                  INTEGER RCODE)

      SUBROUTINE NCAPTC (INTEGER NCID, INTEGER VARID,
     +                   CHARACTER*(*) ATTNAM, INTEGER ATTYPE,
     +                   INTEGER LENSTR, CHARACTER*(*) STRING,
     +                   INTEGER RCODE)
```

There are two FORTRAN subroutines, `NCAPT` and `NCAPTC`, for creating attributes. The first is for attributes of numeric type, and the second is for attributes of character-string type.

`NCID` NetCDF ID, returned from a previous call to `NCOPN` or `NCCRE`.

`VARID` Variable ID.

`ATTNAM` Attribute name. Must begin with an alphabetic character, followed by zero or more alphanumeric characters including the underscore ('_'). Case is significant. Attribute name conventions are assumed by some netCDF generic applications, e.g., `units` as the name for a string attribute that gives the units for a netCDF variable. A table of conventional attribute names is presented in the earlier chapter on the netCDF interface.

`ATTYPE` One of the set of predefined netCDF data types. The valid netCDF data types are `NCBYTE`, `NCCHAR`, `NCSHORT`, `NCLONG`, `NCFLOAT`, and `NCDOUBLE`. For `NCAPTC`, this should always be `NCCHAR` (a blemish in the interface, but required for backward compatibility).

`ATTLEN` In `NCAPT`, the number of numeric values provided for the attribute.

`VALUE` In `NCAPT`, an array of `ATTLEN` data values. The data should be of the appropriate type for the netCDF attribute. **Warning: neither the compiler nor the netCDF software can detect if the wrong type of data is used.**

`STRING` In `NCAPTC`, the character-string value of the attribute.

`LENSTR` In `NCAPTC`, the total declared length (in characters) of the `STRING` parameter. Note that this is not necessarily the same as the value returned by the FORTRAN `LEN` function, because an array argument may be provided.

`RCODE` Returned error code. If no errors occurred, 0 is returned.

Here is an example using `NCAPT` to add a variable attribute named `valid_range` for a netCDF variable named `rh` and a global attribute named `title` to an existing netCDF file named '`foo.nc`':

```
      INCLUDE 'netcdf.inc'
         ...
      INTEGER  NCID, RCODE
      INTEGER  RHID               ! variable ID
      DOUBLE RHRNGE(2)
      DATA RHRNGE /0.0D0, 100.0D0/
         ...
      NCID = NCOPN ('foo.nc', NCWRITE, RCODE)
         ...
      CALL NCREDF (NCID, RCODE)     ! enter define mode
      RHID = NCVID (NCID, 'rh', RCODE)! get ID
         ...
      CALL NCAPT (NCID, RHID, 'valid_range', NCDOUBLE, 2,
     +            RHRNGE, RCODE)
      CALL NCAPTC (NCID, NCGLOBAL, 'title', NCCHAR, 19,
     +            'example netCDF file', RCODE)
         ...
      CALL NCENDF (NCID, RCODE)     ! leave define mode
```

8.3 Get Information about an Attribute: ncattinq and NCAINQ

The function `ncattinq` (or `NCAINQ` for FORTRAN) returns information about a netCDF attribute, given its variable ID and name. The information returned is the type and length of the attribute.

In case of an error, `ncattinq` returns -1; `NCAINQ` returns a nonzero value in `rcode`. Possible causes of errors include:

- The variable ID is invalid for the specified netCDF file.
- The specified attribute does not exist.
- The specified netCDF ID does not refer to an open netCDF file.

C Interface: ncattinq

```
int ncattinq(int ncid, int varid, const char* name,
             nc_type* datatype, int* len);
```

`ncid` NetCDF ID, returned from a previous call to `ncopen` or `nccreate`.

`varid` Variable ID of the attribute's variable, or `NC_GLOBAL` for a global attribute.

`name` Attribute name.

`datatype` Returned attribute type, one of the set of predefined netCDF data types. The type of this parameter, `nc_type`, is defined in the netCDF header file. The valid netCDF data types are `NC_BYTE`, `NC_CHAR`, `NC_SHORT`, `NC_LONG`, `NC_FLOAT`, and `NC_DOUBLE`. If this parameter is given as '`0`' (a null pointer), no type will be returned so no variable to hold the type needs to be declared.

`len` Returned number of values currently stored in the attribute. If the attribute is of type `NC_CHAR`, this is one more than the string length (since the terminating zero byte is stored). If this parameter is given as '`0`' (a null pointer), no length will be returned so no variable to hold this information needs to be declared.

Here is an example using `ncattinq` to find out the type and length of a variable attribute named `valid_range` for a netCDF variable named `rh` and a global attribute named `title` in an existing netCDF file named '`foo.nc`':

```
#include <netcdf.h>
   ...
int  ncid;                  /* netCDF ID */
int  rh_id;                 /* variable ID */
nc_type vr_type, t_type;    /* attribute types */
int  vr_len, t_len;         /* attribute lengths *'

   ...
ncid = ncopen("foo.nc", NC_NOWRITE);
   ...
rh_id = ncvarid (ncid, "rh");
   ...
ncattinq (ncid, rh_id, "valid_range", &vr_type, &vr_len);
ncattinq (ncid, NC_GLOBAL, "title", &t_type, &t_len);
   ...
```

FORTRAN Interface: NCAINQ

```
      SUBROUTINE NCAINQ (INTEGER NCID, INTEGER VARID,
     +                   CHARACTER*(*) ATTNAM, INTEGER ATTYPE,
     +                   INTEGER ATTLEN,INTEGER RCODE)
```

`NCID`	NetCDF ID, returned from a previous call to `NCOPN` or `NCCRE`.
`VARID`	Variable ID of the attribute's variable, or `NCGLOBAL` for a global attribute.
`ATTNAM`	Attribute name.
`ATTYPE`	Returned attribute type, one of the set of predefined netCDF data types. The valid netCDF data types are `NCBYTE`, `NCCHAR`, `NCSHORT`, `NCLONG`, `NCFLOAT`, and `NCDOUBLE`.
`ATTLEN`	Returned number of values currently stored in the attribute. For a string-valued attribute, this is the number of characters in the string.
`RCODE`	Returned error code. If no errors occurred, 0 is returned.

Here is an example using `NCAINQ` to add a variable attribute named `valid_range` for a netCDF variable named `rh` and a global attribute named `title` to an existing netCDF file named '`foo.nc`':

```
 INCLUDE 'netcdf.inc'
    ...
 INTEGER  NCID, RCODE
 INTEGER  RHID                ! variable ID
 INTEGER  VRTYPE, TTYPE       ! attribute types
 INTEGER  VRLEN, TLEN         ! attribute lengths
    ...
 NCID = NCOPN ('foo.nc', NCNOWRIT, RCODE)
    ...
 RHID = NCVID (NCID, 'rh', RCODE)! get ID
    ...
 CALL NCAINQ (NCID, RHID, 'valid_range', VRTYPE, VRLEN,
+             RCODE)
 CALL NCAINQ (NCID, NCGLOBAL, 'title', TTYPE, TLEN,
+             RCODE)
```

8.4 Get Attribute's Values: ncattget and NCAGT(C)

The function `ncattget` (or `NCAGT` or `NCAGTC` for FORTRAN) gets the value(s) of a netCDF attribute, given its variable ID and name.

In case of an error, `ncattget` returns -1; `NCAGT` returns a nonzero value in `rcode`. Possible causes of errors include:

- The variable ID is invalid for the specified netCDF file.
- The specified attribute does not exist.
- The specified netCDF ID does not refer to an open netCDF file.

C Interface: ncattget

```
int ncattget(int ncid, int varid, const char* name, void* value);
```

`ncid` NetCDF ID, returned from a previous call to `ncopen` or `nccreate`.

`varid` Variable ID of the attribute's variable, or `NC_GLOBAL` for a global attribute.

`name` Attribute name.

`value` Returned attribute values. All elements of the vector of attribute values are returned, so you must allocate enough space to hold them. If you don't know how much space to reserve, call `ncattinq` first to find out the length of the attribute.

Here is an example using `ncattget` to determine the values of a variable attribute named `valid_range` for a netCDF variable named `rh` and a global attribute named `title` in an existing netCDF file named '`foo.nc`'. In this example, it is assumed that we don't know how many values will be returned, but that we do know the types of the attributes. Hence, to allocate enough space to store them, we must first inquire about the length of the attributes.

```
#include <netcdf.h>
   ...
int  ncid;                 /* netCDF ID */
int  rh_id;                /* variable ID */
nc_type vr_type, t_type;   /* attribute types */
int  vr_len, t_len;        /* attribute lengths */
double *vr_val;            /* ptr to attribute values */
char *title;               /* ptr to attribute values */
extern char *malloc();     /* memory allocator */

   ...
ncid = ncopen("foo.nc", NC_NOWRITE);
   ...
rh_id = ncvarid (ncid, "rh");
   ...
/* find out how much space is needed for attribute values */
ncattinq (ncid, rh_id, "valid_range", &vr_type, &vr_len);
ncattinq (ncid, NC_GLOBAL, "title", &t_type, &t_len);

/* allocate required space before retrieving values */
vr_val = (double *) malloc(vr_len * nctypelen(vr_type));
title = (char *) malloc(t_len * nctypelen(t_type));

/* get attribute values */
ncattget(ncid, rh_id, "valid_range", vr_val);
ncattget(ncid, NC_GLOBAL, "title", title);
   ...
```

FORTRAN Interface: NCAGT, NCAGTC

```
      SUBROUTINE NCAGT (INTEGER NCID, INTEGER VARID,
     +                  CHARACTER*(*) ATTNAM, type VALUES,
     +                  INTEGER RCODE)
```

```
      SUBROUTINE NCAGTC (INTEGER NCID, INTEGER VARID,
     +                  CHARACTER*(*) ATTNAM, CHARACTER*(*) STRING,
     +                  INTEGER LENSTR, INTEGER RCODE)
```

There are two FORTRAN subroutines, `NCAGT` and `NCAGTC`, for retrieving attribute values. The first is for attributes of numeric type, and the second is for attributes of character-string type.

`NCID`	NetCDF ID, returned from a previous call to `NCOPN` or `NCCRE`.
`VARID`	Variable ID of the attribute's variable, or `NCGLOBAL` for a global attribute.
`ATTNAM`	Attribute name.
`VALUES`	Returned attribute values. All elements of the vector of attribute values are returned, so you must provide enough space to hold them. If you don't know how much space to reserve, call `NCAINQ` first to find out the length of the attribute. **Warning: neither the compiler nor the netCDF software can detect if the wrong type of data is used.**
`STRING`	In `NCAGTC`, the character-string value of the attribute.
`LENSTR`	In `NCAGTC`, the total declared length (in characters) of the `STRING` parameter in the caller. Note that this is not necessarily the same as the value returned by the FORTRAN `LEN` function, because an array argument may be provided. `NCAGTC` will check to make sure the requested data will fit in `LENSTR` characters.
`RCODE`	Returned error code. If no errors occurred, 0 is returned.

Here is an example using `NCAGT` to determine the values of an attribute named `valid_range` for a netCDF variable named `rh` and a global attribute named `title` in an existing netCDF file named '`foo.nc`'. In this example, it is assumed that we don't know how many values will be returned, so we first inquire about the length of the attributes to make sure we have enough space to store them:

```
      INCLUDE 'netcdf.inc'
         ...
      PARAMETER (MVRLEN=3) ! max number of "valid_range" values
      PARAMETER (MTLEN=80) ! max length of "title" attribute
      INTEGER  NCID, RCODE
      INTEGER  RHID              ! variable ID
      INTEGER  VRTYPE, TTYPE     ! attribute types
      INTEGER  VRLEN, TLEN       ! attribute lengths
      DOUBLE PRECISION VRVAL(MVRLEN) ! vr attribute values
      CHARACTER*80 TITLE         ! title attribute values
         ...
      NCID = NCOPN ('foo.nc', NCWRITE, RCODE)
         ...
      RHID = NCVID (NCID, 'rh', RCODE) ! get ID
         ...
* find out attribute lengths, to make sure we have enough space
```

```
      CALL NCAINQ (NCID, RHID, 'valid_range', VRTYPE, VRLEN,
     +             RCODE)
      CALL NCAINQ (NCID, NCGLOBAL, 'title', TTYPE, TLEN,
     +             RCODE)
* get attribute values, if not too big
      IF (VRLEN .GT. MVRLEN) THEN
          WRITE (*,*) 'valid_range attribute too big!'
          CALL EXIT
      ELSE
          CALL NCAGT (NCID, RHID, 'valid_range', VRVAL, RCODE)
      ENDIF
      IF (TLEN .GT. MTLEN) THEN
          WRITE (*,*) 'title attribute too big!'
          CALL EXIT
      ELSE
          CALL NCAGTC (NCID, NCGLOBAL, 'title', TITLE, MTLEN, RCODE)
      ENDIF
```

8.5 Copy Attribute from One NetCDF to Another: ncattcopy and NCACPY

The function `ncattcopy` (or `NCACPY` for FORTRAN) copies an attribute from one open netCDF file to another. It can also be used to copy an attribute from one variable to another within the same netCDF.

In case of an error, `ncattcopy` returns -1; `NCACPY` returns a nonzero value in `rcode`. Possible causes of errors include:

- The input or output variable ID is invalid for the specified netCDF file.
- The specified attribute does not exist.
- The output netCDF is not in define mode and the attribute is new for the output file is larger than the existing attribute.
- The input or output netCDF ID does not refer to an open netCDF file.

C Interface: ncattcopy

```
int ncattcopy(int incdf, int invar, const char* name, int outcdf, int outvar);
```

`incdf`	The netCDF ID of an input netCDF file from which the attribute will be copied, returned from a previous call to `ncopen` or `nccreate`.
`invar`	ID of the variable in the input netCDF file from which the attribute will be copied, or `NC_GLOBAL` for a global attribute.
`name`	Name of the attribute in the input netCDF file to be copied.
`outcdf`	The netCDF ID of the output netCDF file to which the attribute will be copied, returned from a previous call to `ncopen` or `nccreate`. It is permissible for the input

and output netCDF IDs to be the same. The output netCDF file should be in define mode if the attribute to be copied does not already exist for the target variable, or if it would cause an existing target attribute to grow.

`outvar` ID of the variable in the output netCDF file to which the attribute will be copied, or `NC_GLOBAL` to copy to a global attribute.

Here is an example using `ncattcopy` to copy the variable attribute `units` from the variable `rh` in an existing netCDF file named '`foo.nc`' to the variable `avgrh` in another existing netCDF file named '`bar.nc`', assuming that the variable `avgrh` already exists, but does not yet have a `units` attribute:

```
#include <netcdf.h>
   ...
int  ncid1, ncid2;       /* netCDF IDs */
int  rh_id, avgrh_id;      /* variable IDs */
   ...
ncid1 = ncopen("foo.nc", NC_NOWRITE);
ncid2 = ncopen("bar.nc", NC_WRITE);
   ...
rh_id = ncvarid (ncid1, "rh");
avgrh_id = ncvarid (ncid2, "avgrh");
   ...
ncredef(ncid2);            /* enter define mode */
/* copy variable attribute from "rh" to "avgrh" */
ncattcopy(ncid1, rh_id, "units", ncid2, avgrh_id);
   ...
ncendef(ncid2);            /* leave define mode */
```

FORTRAN Interface: NCACPY

```
      SUBROUTINE NCACPY (INTEGER INCDF, INTEGER INVAR,
     +                   CHARACTER*(*) ATTNAM, INTEGER OUTCDF,
     +                   INTEGER OUTVAR, INTEGER RCODE)
```

`INCDF` The netCDF ID of an input netCDF file from which the attribute will be copied, returned from a previous call to `NCOPN` or `NCCRE`.

`INVAR` ID of the variable in the input netCDF file from which the attribute will be copied, or `NCGLOBAL` for a global attribute.

`ATTNAM` Name of the attribute in the input netCDF file to be copied.

`OUTCDF` The netCDF ID of the output netCDF file to which the attribute will be copied, returned from a previous call to `NCOPN` or `NCCRE`. It is permissible for the input and output netCDF IDs to be the same. The output netCDF file should be in define mode if the attribute to be copied does not already exist for the target variable, or if it would cause an existing target attribute to grow.

`OUTVAR` ID of the variable in the output netCDF file to which the attribute will be copied, or `NCGLOBAL` to copy to a global attribute.

Here is an example using `NCACPY` to copy the variable attribute `units` from the variable `rh` in an existing netCDF file named '`foo.nc`' to the variable `avgrh` in another existing netCDF file named '`bar.nc`', assuming that the variable `avgrh` already exists, but does not yet have a `units` attribute:

```
      INCLUDE 'netcdf.inc'
         ...
      INTEGER  NCID1, NCID2     ! netCDF IDs
      INTEGER  RHID, AVRHID       ! variable IDs
         ...
      NCID1 = NCOPN ('foo.nc', NCNOWRIT, RCODE)
      NCID2 = NCOPN ('bar.nc', NCWRITE, RCODE)
         ...
      RHID = NCVID (NCID1, 'rh', RCODE)
      AVRHID = NCVID (NCID2, 'avgrh', RCODE)
         ...
      CALL NCREDF (NCID2, RCODE)  ! enter define mode
* copy variable attribute from "rh" to "avgrh"
      CALL NCACPY (NCID1, RHID, 'units', NCID2, AVRHID, RCODE)
         ...
      CALL NCENDF (NCID2, RCODE)  ! leave define mode
```

8.6 Get Name of Attribute from Its Number: ncattname and NCANAM

The function `ncattname` (or `NCANAM` for FORTRAN) gets the name of an attribute, given its variable ID and number. This function is useful in generic applications that need to get the names of all the attributes associated with a variable, since attributes are accessed by name rather than number in all other attribute functions. The number of an attribute is more volatile than the name, since it can change when other attributes of the same variable are deleted. This is why an attribute number is not called an attribute ID.

In case of an error, `ncattname` returns -1; `NCANAM` returns a nonzero value in `rcode`. Possible causes of errors include:

- The specified variable ID is not valid.
- The specified attribute number is negative or more than the number of attributes defined for the specified variable.
- The specified attribute does not exist.
- The specified netCDF ID does not refer to an open netCDF file.

C Interface: ncattname

```
int ncattname (int ncid, int varid, int attnum, char* name);
```

`ncid` NetCDF ID, returned from a previous call to `ncopen` or `nccreate`.

`varid` ID of the attribute's variable, or `NC_GLOBAL` for a global attribute.

`attnum` Number of the attribute. The attributes for each variable are numbered from 0 (the first attribute) to `nvatts-1`, where `nvatts` is the number of attributes for the variable, as returned from a call to `ncvarinq`.

`name` Returned attribute name. The caller must allocate space for the returned name. The maximum possible length, in characters, of an attribute name is given by the predefined constant `MAX_NC_NAME`. If the name parameter is given as `0` (a null pointer), no name will be returned and no space needs to be allocated.

Here is an example using `ncattname` to determine the name of the first attribute of the variable `rh` in an existing netCDF file named '`foo.nc`':

```
#include <netcdf.h>
   ...
int  ncid;        /* netCDF ID */
int  rh_id;       /* variable ID */
char attname[MAX_NC_NAME];  /* maximum-size attribute name */
   ...
ncid = ncopen("foo.nc", NC_NOWRITE);
   ...
rh_id = ncvarid (ncid, "rh");
   ...
/* get name of first attribute (number 0) */
ncattname(ncid, rh_id, 0, attname);
```

FORTRAN Interface: NCANAM

```
      SUBROUTINE NCANAM (INTEGER NCID, INTEGER VARID,
     +                   INTEGER ATTNUM, CHARACTER*(*) ATTNAM,
     +                   INTEGER RCODE)
```

`NCID` NetCDF ID, returned from a previous call to `NCOPN` or `NCCRE`.

`VARID` ID of the attribute's variable, or `NCGLOBAL` for a global attribute.

`ATTNUM` Number of the attribute. The attributes for each variable are numbered from 1 (the first attribute) to `NVATTS`, where `NVATTS` is the number of attributes for the variable, as returned from a call to `NCVINQ`.

`ATTNAM` Returned attribute name. The caller must allocate space for the returned name. The maximum possible length, in characters, of an attribute name is given by the predefined constant `MAXNCNAM`.

`RCODE` Returned error code. If no errors occurred, 0 is returned.

Here is an example using `NCANAM` determine the name of the first attribute of the variable `rh` in an existing netCDF file named '`foo.nc`':

```
      INCLUDE 'netcdf.inc'
         ...
      INTEGER  NCID      ! netCDF ID
      INTEGER  RHID       ! variable ID
* 31 in the following should be MAXNCNAM
      CHARACTER*31 ATTNAM
         ...
      NCID = NCOPN ('foo.nc', NCNOWRIT, RCODE)
         ...
      RHID = NCVID (NCID, 'rh', RCODE)
         ...
* get name of first attribute (number 1)
      CALL NCANAM (NCID, RHID, 1, ATTNAM, RCODE)
```

8.7 Rename an Attribute: ncattrename and NCAREN

The function `ncattrename` (or `NCAREN` for FORTRAN) changes the name of an attribute. If the new name is longer than the original name, the netCDF file must be in define mode. You cannot rename an attribute to have the same name as another attribute of the same variable.

In case of an error, `ncattrename` returns -1; `NCAREN` returns a nonzero value in `rcode`. Possible causes of errors include:

- The specified variable ID is not valid.
- The new attribute name is already in use for another attribute of the specified variable.
- The specified netCDF file is in data mode and the new name is longer than the old name.
- The specified attribute does not exist.
- The specified netCDF ID does not refer to an open netCDF file.

C Interface: ncattrename

```
int ncattrename (int ncid, int varid, const char* name, const char* newname);
```

`ncid` NetCDF ID, returned from a previous call to `ncopen` or `nccreate`

`varid` ID of the attribute's variable, or `NC_GLOBAL` for a global attribute

`name` The original attribute name.

`newname` The new name to be assigned to the specified attribute. If the new name is longer than the old name, the netCDF file must be in define mode.

Here is an example using `ncattrename` to rename the variable attribute `units` to `Units` for a variable `rh` in an existing netCDF file named '`foo.nc`':

```
#include <netcdf.h>
   ...
int  ncid;        /* netCDF ID */
int  rh_id;       /* variable id */
   ...
ncid = ncopen("foo.nc", NC_NOWRITE);
   ...
rh_id = ncvarid (ncid, "rh");
   ...
/* rename attribute */
ncattrename(ncid, rh_id, "units", "Units");
```

FORTRAN Interface: NCAREN

```
      SUBROUTINE NCAREN (INTEGER NCID, INTEGER VARID,
     +                   CHARACTER*(*) ATTNAM,
     +                   CHARACTER*(*) NEWNAM, INTEGER RCODE)
```

`NCID` NetCDF ID, returned from a previous call to `NCOPN` or `NCCRE`

`VARID` ID of the attribute's variable, or `NCGLOBAL` for a global attribute

`ATTNAM` The original attribute name.

`NEWNAM` The new name to be assigned to the specified attribute. If the new name is longer than the old name, the netCDF file must be in define mode.

`RCODE` Returned error code. If no errors occurred, 0 is returned.

Here is an example using `NCAREN` to rename the variable attribute `units` to `Units` for a variable `rh` in an existing netCDF file named '`foo.nc`':

```
      INCLUDE "netcdf.inc"
         ...
      INTEGER  NCID     ! netCDF ID
      INTEGER  RHID      ! variable ID
         ...
      NCID = NCOPN ("foo.nc", NCNOWRIT, RCODE)
         ...
      RHID = NCVID (NCID, "rh", RCODE)
         ...
* rename attribute
      CALL NCAREN (NCID, RHID, "units", "Units", RCODE)
```

8.8 Delete an Attribute: ncattdel and NCADEL

The function `ncattdel` (or `NCADEL` for FORTRAN) deletes a netCDF attribute from an open netCDF file. The netCDF file must be in define mode.

In case of an error, `ncattdel` returns -1; `NCADEL` returns a nonzero value in `rcode`. Possible causes of errors include:

- The specified variable ID is not valid.
- The specified netCDF file is in data mode.
- The specified attribute does not exist.
- The specified netCDF ID does not refer to an open netCDF file.

C Interface: ncattdel

```
int ncattdel (int ncid, int varid, const char* name);
```

`ncid`	NetCDF ID, returned from a previous call to `ncopen` or `nccreate`.
`varid`	ID of the attribute's variable, or `NC_GLOBAL` for a global attribute.
`name`	The name of the attribute to be deleted.

Here is an example using `ncattdel` to delete the variable attribute `Units` for a variable `rh` in an existing netCDF file named '`foo.nc`':

```
#include <netcdf.h>
   ...
int  ncid;        /* netCDF ID */
int  rh_id;       /* variable ID */
   ...
ncid = ncopen("foo.nc", NC_WRITE);
   ...
rh_id = ncvarid (ncid, "rh");
   ...
/* delete attribute */
ncredef(ncid);                  /* enter define mode */
ncattdel(ncid, rh_id, "Units");
ncendef(ncid);                  /* leave define mode */
```

FORTRAN Interface: NCADEL

```
      SUBROUTINE NCADEL (INTEGER NCID, INTEGER VARID,
     +                   CHARACTER*(*) ATTNAM, INTEGER RCODE)
```

`NCID`	NetCDF ID, returned from a previous call to `NCOPN` or `NCCRE`.
`VARID`	ID of the attribute's variable, or `NCGLOBAL` for a global attribute.
`ATTNAM`	The original attribute name.

`RCODE` Returned error code. If no errors occurred, 0 is returned.

Here is an example using `NCADEL` to delete the variable attribute `Units` for a variable `rh` in an existing netCDF file named '`foo.nc`':

```
           INCLUDE 'netcdf.inc'
              ...
           INTEGER  NCID     ! netCDF ID
           INTEGER  RHID      ! variable ID
              ...
           NCID = NCOPN ('foo.nc', NCWRITE, RCODE)
              ...
           RHID = NCVID (NCID, 'rh', RCODE)
              ...
     * delete attribute
           CALL NCREDF (NCID, RCODE)  ! enter define mode
           CALL NCADEL (NCID, RHID, 'Units', RCODE)
           CALL NCENDF (NCID, RCODE)  ! leave define mode
```

9 NetCDF File Structure and Performance

NetCDF is a data abstraction for array-oriented data access and a software library that provides a concrete implementation of the interfaces that support that abstraction. The implementation provides a machine-independent format for representing arrays. Although the netCDF file format is hidden below the interfaces, some understanding of the implementation and associated file structure may help to make clear which netCDF operations are expensive and why.

For a detailed description of the netCDF *format*, see Appendix B [File Format Specification], page 143. It is not needed to read and write netCDF files or understand efficiency issues. Programs that use only the documented interfaces and that make no other assumptions about the format will continue to work even if the netCDF format is changed in the future, because any such change will be made below the documented interfaces and will support earlier versions of netCDF data.

This chapter describes the structure of a netCDF file and some characteristics of the XDR layer that provides network transparency in enough detail to understand netCDF performance issues.

9.1 Parts of a NetCDF File

A netCDF dataset is stored as a single file comprising two parts:

- a *header*, containing all the information about dimensions, attributes, and variables except for the variable data;
- a *data* part, comprising *fixed-size data*, containing the data for variables that don't have an unlimited dimension; and *record data*, containing the data records for variables that have an unlimited dimension.

All the data are represented in XDR form to make them machine-independent.

The descriptive header at the beginning of the netCDF file is an XDR encoding of a high-level data structure that represents information about the dimensions, variables, and attributes in the file. The variable descriptions in this header contain offsets to the beginning of each variable's data or the relative offset of a variable within a record. The descriptions also contain the dimension size and information needed to determine how to map multidimensional indices for each variable to the appropriate offsets.

This header has no usable extra space; it is only as large as it needs to be for the dimensions, variables, and attributes in each netCDF file. This has the advantage that netCDF files are compact, requiring very little overhead to store the ancillary data that makes the files self-describing. A potential disadvantage of this organization is that any operation on a netCDF file that requires expanding the header, for example adding new dimensions and new variables to an existing netCDF file, will be as expensive as copying the file. This expense is incurred when `ncendef()` is called, after a call to `ncredef()`. If you create all necessary dimensions, variables, and attributes *before* writing variable data, and avoid later additions and renamings of netCDF components that require more space in the header part of the file, you avoid the cost associated with expanding the header.

The fixed-size data part that follows the header contains all the variable data for variables that do not employ the unlimited (record) dimension. The data for each variable is stored contiguously in this part of the file. If there is no unlimited dimension, this is the last part of the netCDF file.

The record-data part that follows the fixed-size data consists of a variable number of records, each of which contains data for all the record variables. The record data for each variable is stored contiguously in each record.

The order in which the data in the fixed-size data part and in each record appears is the same as the order in which the variables were defined, in increasing numerical order by netCDF variable ID. This knowledge can sometimes be used to enhance data access performance, since the best data access is currently achieved by reading or writing the data in sequential order.

9.2 The XDR Layer

XDR is a standard for describing and encoding data and a library of functions for external data representation, allowing programmers to encode data structures in a machine-independent way. NetCDF employs XDR for representing all data, in both the header part and the data parts. XDR is used to write portable data that can be read on any other machine for which the XDR library has been implemented.

9.3 General XDR Considerations

Many vendors provide an XDR library along with other C run-time libraries. The netCDF software distribution also includes Sun's portable implementation of XDR for platforms that don't already have a vendor-supplied XDR library.

An I/O layer implemented much like the C standard I/O (stdio) library is used by the XDR layer to read and write XDR-encoded data to netCDF files. Hence an understanding of the standard I/O library provides answers to most questions about multiple processes accessing data concurrently, the use of I/O buffers, and the costs of opening and closing netCDF files. In particular, it is possible to have one process writing a netCDF file while other processes read it. Data reads and writes are no more atomic than calls to stdio `fread()` and `fwrite()`. An `ncsync()` call (`NCSNC()` for FORTRAN) is analogous to the `fflush()` call in the standard I/O library, writing unwritten buffered data so other processes can read it; `ncsync()` also brings header changes up-to-date (e.g., changes to attribute values).

As in the stdio library, flushes are also performed when "seeks" occur to a different area of the file. Hence the order of read and write operations can influence I/O performance significantly. Reading data in the same order in which it was written within each record will minimize buffer flushes.

There is one unusual case where the situation is more complex: when a writer enters define mode to add some additional dimensions, variables, or attributes to an existing netCDF file that is also open for reading by other processes. In this case, when the writer leaves define mode, a new copy of the file is created with the new dimensions, attributes, or variables and the old data, but readers that still have the file open will not see the changes, unless they close and reopen the file.

You should not expect netCDF data access to work with multiple writers having the same file open for writing simultaneously.

It is possible to tune an implementation of netCDF for some platforms by replacing the I/O layer beneath XDR with a different platform-specific I/O layer. This has been done for Crays, for example. This may change the similarities between netCDF and standard I/O, and hence characteristics related to data sharing, buffering, and the cost of I/O operations.

The cost of using a canonical representation for data like XDR varies according to the type of data and whether the XDR form is the same as the machine's native form for that type. XDR is especially efficient for byte, character, and short integer data.

For some data types on some machines, the time required to convert data to and from XDR form can be significant. The best case is byte arrays, for which very little conversion expense occurs, since the XDR library has built-in support for them. The netCDF implementation includes similar support added to XDR for arrays of short (16-bit) integers. The worst case is reading or writing large arrays of floating-point data on a machine that does not use IEEE floating-point as its native representation. The XDR library incurs the expense of a function call for each floating-point quantity accessed. On some architectures the cost of a function invocation for each floating-point number can dominate the cost of netCDF access to floating-point fields.

The distributed netCDF implementation is meant to be portable. Platform-specific ports that further optimize the implementation for better I/O performance or that unroll the loops in the XDR library to optimize XDR conversion of long integer and floating-point arrays are practical and desirable in cases where higher performance for data access is necessary.

9.4 UNICOS Optimization

As was mentioned in the previous section, it is possible to replace the I/O layer that is used by XDR in order to increase I/O efficiency. This has been done for UNICOS, the operating system of Cray computers (e.g. the Cray Y-MP).

Additionally, it is possible for the user to obtain even greater I/O efficiency through appropriate setting of the `NETCDF_FFIOSPEC` environment variable. This variable specifies the Flexible File I/O buffers for netCDF I/O when executing under the UNICOS operating system (the variable is ignored on other operating systems). An appropriate specification can greatly increase the efficiency of netCDF I/O — to the extent that it can rival and actually surpass default FORTRAN binary I/O. Possible specifications include the following:

- `bufa:336:2` 2, asynchronous, I/O buffers of 336 blocks each (i.e. double buffering). This is the default specification and favors sequential I/O.
- `cache:256:8:2` 8, synchronous, 256-block pages with a 2 block read-ahead/write-behind factor. This favors larger random accesses.
- `cachea:256:8:2` 8, asynchronous, 256-block pages with a 2 block read-ahead/write-behind factor. This also favors larger random accesses.
- `cachea:8:256:0` 256, asynchronous, 8-block pages without read-ahead/write-behind. This favors many smaller pages without read-ahead for more random accesses as typified by slicing netCDF arrays.
- `cache:8:256:0,cachea.sds:1024:4:1` This is a two layer cache. The first (synchronous) layer is composed of 256 8-block pages in memory, the second (asynchronous) layer is composed of 4 1024-block pages on the SSD. This scheme works well when accesses proceed through the file in random waves roughly 2x1024-blocks wide.

All of the options/configurations supported in CRI's FFIO library are available through this mechanism. We recommend that you look at CRI's I/O optimization guide for information on using FFIO to it's fullest. This mechanism is also compatible with CRI's EIE I/O library.

Tuning the `NETCDF_FFIOSPEC` variable to a program's I/O pattern can dramatically improve performance. Speedups of two orders of magnitude have been seen.

10 NetCDF Utilities

One of the primary reasons for using the netCDF interface for applications that deal with arrays is to take advantage of higher-level netCDF utilities and generic applications for netCDF data. Currently two netCDF utilities are available as part of the netCDF software distribution: `ncgen` and `ncdump`. Users have contributed other netCDF utilities, and various visualization and analysis packages are available that access netCDF data. For an up-to-date list of freely-available and commercial software that can access or manipulate netCDF data, see the NetCDF Software list (‘`http://www.unidata.ucar.edu/packages/netcdf/software.html`’).

This chapter describes the `ncgen` and `ncdump` utilities. These two tools convert between binary netCDF files and a text representation of netCDF files. The output of `ncdump` and the input to `ncgen` is a text description of a netCDF file in a tiny language known as CDL (network Common data form Description Language).

10.1 CDL Syntax

Below is an example of CDL, describing a netCDF file with several named dimensions (`lat`, `lon`, `time`), variables (`z`, `t`, `p`, `rh`, `lat`, `lon`, `time`), variable attributes (`units`, `_FillValue`, `valid_range`), and some data.

```
netcdf foo {    // example netCDF specification in CDL

dimensions:
lat = 10, lon = 5, time = unlimited ;

variables:
  long    lat(lat), lon(lon), time(time);
  float   z(time,lat,lon), t(time,lat,lon);
  double  p(time,lat,lon);
  long    rh(time,lat,lon);

  lat:units = "degrees_north";
  lon:units = "degrees_east";
  time:units = "seconds";
  z:units = "meters";
  z:valid_range = 0., 5000.;
  p:_FillValue = -9999.;
  rh:_FillValue = -1;

data:
  lat   = 0, 10, 20, 30, 40, 50, 60, 70, 80, 90;
  lon   = -140, -118, -96, -84, -52;
}
```

All CDL statements are terminated by a semicolon. Spaces, tabs, and newlines can be used freely for readability. Comments may follow the double slash characters `//` on any line.

A CDL description consists of three optional parts: dimensions, variables, and data. The variable part may contain variable declarations and attribute assignments.

A dimension is used to define the shape of one or more of the multidimensional variables described by the CDL description. A dimension has a name and a size. At most one dimension in a CDL description can have the unlimited size, which means a variable using this dimension can grow to any length (like a record number in a file).

A variable represents a multidimensional array of values of the same type. A variable has a name, a data type, and a shape described by its list of dimensions. Each variable may also have associated attributes (see below) as well as data values. The name, data type, and shape of a variable are specified by its declaration in the variable section of a CDL description. A variable may have the same name as a dimension; by convention such a variable contains coordinates of the dimension it names.

An attribute contains information about a variable or about the whole netCDF dataset. Attributes may be used to specify such properties as units, special values, maximum and minimum valid values, and packing parameters. Attribute information is represented by single values or arrays of values. For example, `units` is an attribute represented by a character array such as `celsius`. An attribute has an associated variable, a name, a data type, a length, and a value. In contrast to variables that are intended for data, attributes are intended for ancillary data (data about data).

In CDL, an attribute is designated by a variable and attribute name, separated by a colon ('`:`'). It is possible to assign global attributes to the netCDF file as a whole by omitting the variable name and beginning the attribute name with a colon ('`:`'). The data type of an attribute in CDL is derived from the type of the value assigned to it. The length of an attribute is the number of data values or the number of characters in the character string assigned to it. Multiple values are assigned to non-character attributes by separating the values with commas ('`,`'). All values assigned to an attribute must be of the same type.

CDL names for variables, attributes, and dimensions may be any combination of alphabetic or numeric characters as well as '`_`' and '`-`' characters, but names beginning with '`_`' are reserved for use by the library. Case is significant in CDL names. The netCDF library does not enforce any restrictions on netCDF names, so it is possible (though unwise) to define variables with names that are not valid CDL names. The names for the primitive data types are reserved words in CDL, so the names of variables, dimensions, and attributes must not be type names.

The optional data section of a CDL description is where netCDF variables may be initialized. The syntax of an initialization is simple:

variable = *value_1*, *value_2*, ...;

The comma-delimited list of constants may be separated by spaces, tabs, and newlines. For multidimensional arrays, the last dimension varies fastest. Thus, row-order rather than column order is used for matrices. If fewer values are supplied than are needed to fill a variable, it is extended with the fill value. The types of constants need not match the type declared for a variable; coercions are done to convert integers to floating point, for example. All meaningful type conversions are supported.

A special notation for fill values is supported: the '`_`' character designates a fill value for variables.

10.2 CDL Data Types

The CDL data types are:

`char`	Characters.
`byte`	Eight-bit integers.
`short`	16-bit signed integers.
`long`	32-bit signed integers.
`int`	(Synonymous with long).
`float`	IEEE single-precision floating point (32 bits).
`real`	(Synonymous with float).
`double`	IEEE double-precision floating point (64 bits).

Except for the added data-type `byte` and the lack of the type qualifier `unsigned`, CDL supports the same primitive data types as C. In declarations, type names may be specified in either upper or lower case.

The `byte` type differs from the `char` type in that it is intended for eight-bit data, and the zero byte has no special significance, as it may for character data. The `ncgen` utility converts `byte` declarations to `char` declarations in the output C code and to `BYTE`, `INTEGER*1`, or similar platform-specific declaration in output FORTRAN code.

The `short` type holds values between -32768 and 32767. The `ncgen` utility converts `short` declarations to `short` declarations in the output C code and to `INTEGER*2` declaration in output FORTRAN code.

The `long` type can hold values between -2147483648 and 2147483647. The `ncgen` utility converts `long` declarations to `nclong` declarations in the output C code and to `INTEGER` declarations in output FORTRAN code. In CDL declarations `int` and `integer` are accepted as synonyms for `long`.

The `float` type can hold values between about -3.4+38 and 3.4+38, with external representation as 32-bit IEEE normalized single-precision floating-point numbers. The `ncgen` utility converts `float` declarations to `float` declarations in the output C code and to `REAL` declarations in output FORTRAN code. In CDL declarations `real` is accepted as a synonym for `float`.

The `double` type can hold values between about -1.7+308 and 1.7+308, with external representation as 64-bit IEEE standard normalized double-precision, floating-point numbers. The `ncgen` utility converts `double` declarations to `double` declarations in the output C code and to `DOUBLE PRECISION` declarations in output FORTRAN code.

10.3 CDL Notation for Data Constants

This section describes the CDL notation for constants.

Attributes are initialized in the `variables` section of a CDL description by providing a list of constants that determines the attribute's type and length. (In the C and FORTRAN procedural interfaces to the netCDF library, the type and length of an attribute must be explicitly provided when it is defined.) CDL defines a syntax for constant values that permits distinguishing among different netCDF types. The syntax for CDL constants is similar to C syntax, except that type suffixes are appended to `short`s and `float`s to distinguish them from `long`s and `double`s.

A `byte` constant is represented by a single character or multiple character escape sequence enclosed in single quotes. For example:

```
'a'     // ASCII a
'\0'    // a zero byte
'\n'    // ASCII newline character
'\33'   // ASCII escape character (33 octal)
'\x2b'  // ASCII plus (2b hex)
'\376'  // 377 octal = -127 (or 254) decimal
```

Character constants are enclosed in double quotes. A character array may be represented as a string enclosed in double quotes. Multiple strings are concatenated into a single array of characters, permitting long character arrays to appear on multiple lines. To support multiple variable-length string values, a conventional delimiter such as '`,`' may be used, but interpretation of any such convention for a string delimiter must be implemented in software above the netCDF library layer. The usual escape conventions for C strings are honored. For example:

```
"a"              // ASCII 'a'
"Two\nlines\n"   // a 10-character string with two embedded newlines
"a bell:\007"    // a string containing an ASCII bell
"ab","cde"       // the same as "abcde"
```

The form of a `short` constant is an integer constant with an '`s`' or '`S`' appended. If a `short` constant begins with '`0`', it is interpreted as octal. When it begins with '`0x`', it is interpreted as a hexadecimal constant. For example:

```
2s      // a short 2
0123s   // octal
0x7ffs  // hexadecimal
```

The form of a `long` constant is an ordinary integer constant, although it is acceptable to append an optional '`l`' or '`L`'. If a `long` constant begins with '`0`', it is interpreted as octal. When it begins with '`0x`', it is interpreted as a hexadecimal constant. Examples of valid `long` constants include:

```
-2
1234567890L
0123            // octal
0x7ff           // hexadecimal
```

The `float` type is appropriate for representing data with about seven significant digits of precision. The form of a `float` constant is the same as a C floating-point constant with an '`f`' or '`F`' appended. A decimal point is required in a CDL `float` to distinguish it from an integer. For example, the following are all acceptable `float` constants:

```
-2.0f
3.14159265358979f       // will be truncated to less precision
1.f
.1f
```

The `double` type is appropriate for representing floating-point data with about 16 significant digits of precision. The form of a `double` constant is the same as a C floating-point constant. An optional '`d`' or '`D`' may be appended. A decimal point is required in a CDL `double` to distinguish it from an `integer`. For example, the following are all acceptable double constants:

```
-2.0
3.141592653589793
1.0e-20
1.d
```

10.4 ncgen

The `ncgen` tool generates a netCDF file or a C or FORTRAN program that creates a netCDF file. If no options are specified in invoking `ncgen`, the program merely checks the syntax of the CDL input, producing error messages for any violations of CDL syntax.

UNIX syntax for invoking `ncgen`:

`ncgen [-b] [-o` *netcdf-file*`] [-c] [-f] [-n] [`*input-file*`]`

where:

'`-b`' Create a (binary) netCDF file. If the '`-o`' option is absent, a default file name will be constructed from the netCDF name (specified after the `netcdf` keyword in the input) by appending the '`.nc`' extension. **Warning: if a file already exists with the specified name it will be overwritten.**

'`-o netcdf-file`'
: Name for the netCDF file created. If this option is specified, it implies the '`-b`' option. (This option is necessary because netCDF files are direct-access files created with seek calls, and hence cannot be written to standard output.)

'`-c`'
: Generate C source code that will create a netCDF file matching the netCDF specification. The C source code is written to standard output. This is only useful for relatively small CDL files, since all the data is included in variable initializations in the generated program.

'`-f`'
: Generate FORTRAN source code that will create a netCDF file matching the netCDF specification. The FORTRAN source code is written to standard output. This is only useful for relatively small CDL files, since all the data is included in variable initializations in the generated program.

'`-n`'
: Like the '`-b`' option, except creates a netCDF file with the obsolete '`.cdf`' extension instead of the '`.nc`' extension, in the absence of an output filename specified by the '`-o`' option. This option is only supported for backward compatibility.

Examples

Check the syntax of the CDL file '`foo.cdl`':

```
ncgen foo.cdl
```

From the CDL file '`foo.cdl`', generate an equivalent binary netCDF file named '`bar.nc`':

```
ncgen -o bar.nc foo.cdl
```

From the CDL file '`foo.cdl`', generate a C program containing the netCDF function invocations necessary to create an equivalent binary netCDF file:

```
ncgen -c foo.cdl > foo.c
```

10.5 ncdump

The `ncdump` tool generates the CDL text representation of a netCDF file on standard output, optionally excluding some or all of the variable data in the output. The output from `ncdump` is intended to be acceptable as input to `ncgen`. Thus `ncdump` and `ncgen` can be used as inverses to transform data representation between binary and text representations.

`ncdump` may also be used as a simple browser for netCDF data files, to display the dimension names and sizes; variable names, types, and shapes; attribute names and values; and optionally, the values of data for all variables or selected variables in a netCDF file.

`ncdump` defines a default format used for each type of netCDF variable data, but this can be overridden if a `C_format` attribute is defined for a netCDF variable. In this case, `ncdump` will use the `C_format` attribute to format values for that variable. For example, if floating-point data for the netCDF variable `Z` is known to be accurate to only three significant digits, it might be appropriate to use the variable attribute

`ncdump` uses '`_`' to represent data values that are equal to the `_FillValue` attribute for a variable, intended to represent data that has not yet been written. If a variable has no `_FillValue` attribute, the default fill value for the variable type is used unless the variable is of byte type.

```
Z:C_format = "%.3g"
```

UNIX syntax for invoking `ncdump`:

```
ncdump  [ -c | -h]  [-v var1,...]  [-b lang]  [-f lang]
[-l len]  [ -d fdig[,ddig]] [ -n name]  [input-file]
```

where:

'`-c`' Show the values of *coordinate* variables (variables that are also dimensions) as well as the declarations of all dimensions, variables, and attribute values. Data values of non-coordinate variables are not included in the output. This is often the most suitable option to use for a brief look at the structure and contents of a netCDF file.

'`-h`' Show only the *header* information in the output, that is, output only the declarations for the netCDF dimensions, variables, and attributes of the input file, but no data values for any variables. The output is identical to using the '`-c`' option except that the values of coordinate variables are not included. (At most one of '`-c`' or '`-h`' options may be present.)

'`-v var1,...`'
The output will include data values for the specified variables, in addition to the declarations of all dimensions, variables, and attributes. One or more variables must be specified by name in the comma-delimited list following this option. The list must be a single argument to the command, hence cannot contain blanks or other white space characters. The named variables must be valid netCDF variables in the input-file. The default, without this option and in the absence of the '`-c`' or '`-h`' options, is to include data values for *all* variables in the output.

'`-b lang`' A brief annotation in the form of a CDL comment (text beginning with the characters '`//`') will be included in the data section of the output for each 'row' of data, to help identify data values for multidimensional variables. If *lang* begins with '`C`' or '`c`', then C language conventions will be used (zero-based indices, last dimension varying fastest). If *lang* begins with '`F`' or '`f`', then FORTRAN language conventions will be used (one-based indices, first dimension varying fastest). In either case, the data will be presented in the same order; only the annotations will differ. This option may be useful for browsing through large volumes of multidimensional data.

'`-f lang`' Full annotations in the form of trailing CDL comments (text beginning with the characters '`//`') for every data value (except individual characters in character arrays) will be included in the data section. If *lang* begins with '`C`' or '`c`', then C language conventions will be used (zero-based indices, last dimension varying fastest). If *lang* begins with '`F`' or '`f`', then FORTRAN language conventions will be used (one-based indices, first dimension varying fastest). In either case, the data will be presented in the same order; only the annotations will differ. This option may be useful for piping data into other filters, since each data value appears on a separate line, fully identified.. (At most one of '`-b`' or '`-f`' options may be present.)

'`-l len`' Changes the default maximum line length (80) used in formatting lists of non-character data values.

'`-d float_digits[,double_digits]`'
Specifies default number of significant digits to use in displaying floating-point or double precision data values for variables that don't have a 'C_format' attribute. Floating-point data will be displayed with *float_digits* significant digits. If *double_digits* is also specified, double-precision values will be displayed with that many significant digits. If a variable has a 'C_format' attribute, that overrides any specified floating-point default. In the absence of any '`-d`' specifications, floating-point and double-precision data are displayed with 7 and 15 significant digits respectively. CDL files can be made smaller if less precision is required. If both floating-point and double-precision precisions are specified, the two values must appear separated by a comma (no blanks) as a single argument to the command.

'`-n name`' CDL requires a name for a netCDF dataset, for use by '`ncgen -b`' in generating a default netCDF file name. By default, `ncdump` constructs this name from the last component of the file name of the input netCDF file by stripping off any extension it has. Use the '`-n`' option to specify a different name. Although the output file name used by '`ncgen -b`' can be specified, it may be wise to have `ncdump` change the default name to avoid inadvertently overwriting a valuable netCDF file when using `ncdump`, editing the resulting CDL file, and using '`ncgen -b`' to generate a new netCDF file from the edited CDL file.

Examples

Look at the structure of the data in the netCDF file '`foo.nc`':

```
ncdump -c foo.nc
```

Produce an annotated CDL version of the structure and data in the netCDF file '`foo.nc`', using C-style indexing for the annotations:

```
ncdump -b c foo.nc > foo.cdl
```

Output data for only the variables `uwind` and `vwind` from the netCDF file '`foo.nc`', and show the floating-point data with only three significant digits of precision:

```
ncdump -v uwind,vwind -d 3 foo.nc
```

Produce a fully-annotated (one data value per line) listing of the data for the variable `omega`, using FORTRAN conventions for indices, and changing the netCDF dataset name in the resulting CDL file to `omega`:

```
ncdump -v omega -f fortran -n omega foo.nc > Z.cdl
```

11 Answers to Some Frequently Asked Questions

This chapter contains answers to some of the most frequently asked questions about netCDF. A more comprehensive and up-to-date FAQ document for netCDF is maintained at

'`http://www.unidata.ucar.edu/packages/netcdf/faq.html`'

What Is netCDF?

NetCDF (network Common Data Form) is an interface for array-oriented data access and a freely-distributed collection of software libraries for C, FORTRAN, C++, and perl that provide implementations of the interface. The netCDF software was developed by Glenn Davis, Russ Rew, and Steve Emmerson at the Unidata Program Center in Boulder, Colorado, and augmented by contributions from other netCDF users. The netCDF libraries define a machine-independent format for representing arrays. Together, the interface, libraries, and format support the creation, access, and sharing of arrays.

NetCDF data is:

- Self-Describing. A netCDF file includes information about the data it contains.
- Network-transparent. A netCDF file is represented in a form that can be accessed by computers with different ways of storing integers, characters, and floating-point numbers.
- Direct-access. A small subset of a large dataset may be accessed efficiently, without first reading through all the preceding data.
- Appendable. Data can be appended to a netCDF dataset along one dimension without copying the dataset or redefining its structure. The structure of a netCDF dataset can be changed, though this sometimes causes the dataset to be copied.
- Sharable. One writer and multiple readers may simultaneously access the same netCDF file.

How do I get the netCDF software package?

Source distributions are available via anonymous FTP from the directory

'`ftp.unidata.ucar.edu:pub/netcdf/`'

Files in that directory include:

`netcdf.tar.Z`
: A compressed tar file of source code for the latest general release.

`netcdf-beta.tar.Z`
: The current beta-test release.

Binary distributions for some platforms are available from the directory '`ftp://ftp.unidata.ucar.edu/pub/binary/`'

Source for the perl interface is available as a separate package, via anonymous FTP from the directory

'`ftp://ftp.unidata.ucar.edu/pub/netcdf-perl/`'

Is there any access to netCDF information on the World Wide Web?

Yes, the latest version of this FAQ document as well as a hypertext version of the NetCDF User's Guide and other information about netCDF are available from

'`http://www.unidata.ucar.edu/packages/netcdf`'

What has changed since the previous release?

Version 2.4 incorporates support for new platforms and updated versions of previously-supported platforms, provides new optimizations for Cray/UNICOS, incorporates fixes for reported bugs, improves the documentation, and improves ease of installation. For more details, see

'`http://www.unidata.ucar.edu/packages/netcdf/release-notes.html`'

Is there a mailing list for netCDF discussions and questions?

Yes. For information about the mailing list and how to subscribe or unsubscribe, send a message to `majordomo@unidata.ucar.edu` with no subject and with the following line in the body of the message:

```
info netcdfgroup
```

Who else uses netCDF?

The netCDF mailing list has almost 500 addresses (some of which are aliases to more addresses) in fifteen countries. Several groups have adopted netCDF as a standard way to represent some forms of array-oriented data, including groups in the atmospheric sciences, hydrology, oceanography, environmental modeling, geophysics, chromatography, mass spectrometry, and neuro-imaging.

A description of some of the projects and groups that have used netCDF is available from

'`http://www.unidata.ucar.edu/packages/netcdf/usage.html`'

What is the physical format for a netCDF files?

See Chapter 9 [NetCDF File Structure and Performance], page 121, for an explanation of the physical structure of netCDF data at a high enough level to make clear the performance implications of different data organizations. See Appendix B [File Format Specification], page 143, for a detailed specification of the file format.

Programs that access netCDF data should perform all access through the documented interfaces, rather than relying on the physical format of netCDF data. That way, any future changes to the format will not require changes to programs, since any such changes will be accompanied by changes in the library to support both the old and new versions of the format.

What does netCDF run on?

The current version of netCDF has been tested successfully on the following platforms:

- AIX-4.1
- HPUX-9.05
- IRIX-5.3
- IRIX64-6.1
- MSDOS (using gcc, f2c, and GNU make)
- OSF1-3.2
- OpenVMS-6.2
- OS/2 2.1
- SUNOS-4.1.4
- SUNOS-5.5
- ULTRIX-4.5
- UNICOS-8
- Windows NT-3.51

What other software is available for netCDF data?

Utilities available in the current netCDF distribution from Unidata are `ncdump`, for converting netCDF files to an ASCII human-readable form, and `ncgen` for converting from the ASCII human-readable form back to a binary netCDF file or a C or FORTRAN program for generating the netCDF file.

Several commercial and freely available analysis and data visualization packages have been adapted to access netCDF data. More information about these packages and other software that can be used to manipulate or display netCDF data is available from

‘`http://www.unidata.ucar.edu/packages/netcdf/software.html`’

What other formats are available for array-oriented data?

The *Scientific Data Format Information FAQ*, available from

'`http://www.cis.ohio-state.edu/hypertext/faq/usenet/sci-data-formats/faq.html`' , provides a good description of other access interfaces and formats for array-oriented data, including CDF and HDF.

Why do netCDF calls just exit on errors instead of returning an error indicator?

The default error handling behavior of all the netCDF functions is to exit on error, but this behavior is under programmer control. You can independently control the fatality of errors and the appearance of messages from errors detected in netCDF library calls. See Section 4.5 [Error Handling], page 32, for more information.

How do I make a bug report?

If you find a bug, send a description to `support@unidata.ucar.edu`. This is also the address to use for questions or discussions about netCDF that are not appropriate for the entire `netcdfgroup` mailing list.

How do I search through past problem reports?

A search form is available at the bottom of the netCDF home page providing a full-text search of the support questions and answers about netCDF provided by Unidata support staff.

How does the C++ interface differ from the C interface?

It provides all the functionality of the C interface (except for the subsampled or mapped array access of `ncvarputg` and `ncvargetg`), improves type safety by eliminating use of `void*` pointers, and is somewhat simpler to use than the C interface. With the C++ interface, no IDs are needed for netCDF components, there is no need to specify types when creating attributes, and less indirection is required for dealing with dimensions. However, the C++ interface is less mature and less-widely used than the C interface, and the documentation for the C++ interface is less extensive, assuming a familiarity with the netCDF data model and the C interface.

How does the FORTRAN interface differ from the C interface?

It provides all the functionality of the C interface. The FORTRAN interface uses FORTRAN conventions for array indices, subscript order, and strings. There is no difference in the on-disk format for data written from the different language interfaces. Data written by a C language program may be read from a FORTRAN program and vice-versa.

How does the Perl interface differ from the C interface?

It provides all the functionality of the C interface. The Perl interface uses Perl conventions for arrays and strings. There is no difference in the on-disk format for data written from the different language interfaces. Data written by a C language program may be read from a Perl program and vice-versa.

Appendix A Units

The Unidata Program Center has developed a units library to convert between formatted and binary forms of units specifications and perform unit algebra on the binary form. Though the units library is self-contained and there is no dependency between it and the netCDF library, it is nevertheless useful in writing generic netCDF programs and we suggest you obtain it. The library and associated documentation is available from '`http://www.unidata.ucar.edu/packages/udunits/`'.

The following are examples of units strings that can be interpreted by the `utScan()` function of the Unidata units library:

```
10 kilogram.meters/seconds2
10 kg-m/sec2
10 kg m/s^2
10 kilogram meter second-2
(PI radian)2
degF
100rpm
geopotential meters
33 feet water
milliseconds since 1992-12-31 12:34:0.1 -7:00
```

A unit is specified as an arbitrary product of constants and unit-names raised to arbitrary integral powers. Division is indicated by a slash '`/`'. Multiplication is indicated by whitespace, a period '`.`', or a hyphen '`-`'. Exponentiation is indicated by an integer suffix or by the exponentiation operators '`^`' and '`**`'. Parentheses may be used for grouping and disambiguation. The timestamp in the last example is handled as a special case.

Arbitrary Galilean transformations (i.e. $y = ax + b$) are allowed. In particular, temperature conversions are correctly handled. The specification:

```
degF @ 32
```

indicates a Fahrenheit scale with the origin shifted to thirty-two degrees Fahrenheit (i.e. to zero Celsius). Thus, the Celsius scale is equivalent to the following unit:

```
1.8 degF @ 32
```

Note that the origin-shift operation takes precedence over multiplication. In order of increasing precedence, the operations are division, multiplication, origin-shift, and exponentiation.

`utScan()` understands all the SI prefixes (e.g. "mega" and "milli") plus their abbreviations (e.g. "M" and "m")

The function utPrint() always encodes a unit specification one way. To reduce misunderstandings, it is recommended that this encoding style be used as the default. In general, a unit is encoded in terms of basic units, factors, and exponents. Basic units are separated by spaces, and any exponent directly appends its associated unit. The above examples would be encoded as follows:

```
10 kilogram meter second-2
9.8696044 radian2
0.555556 kelvin @ 255.372
10.471976 radian second-1
9.80665 meter2 second-2
98636.5 kilogram meter-1 second-2
0.001 seconds since 1992-12-31 19:34:0.1000 UTC
```

(Note that the Fahrenheit unit is encoded as a deviation, in fractional kelvins, from an origin at 255.372 kelvin, and that the time in the last example has been referenced to UTC.)

The database for the units library is a formatted file containing unit definitions and is used to initialize this package. It is the first place to look to discover the set of valid names and symbols.

The format for the units-file is documented internally and the file may be modified by the user as necessary. In particular, additional units and constants may be easily added (including variant spellings of existing units or constants).

`utScan()` is case-sensitive. If this causes difficulties, you might try making appropriate additional entries to the units-file.

Some unit abbreviations in the default units-file might seem counter-intuitive. In particular, note the following:

For	Use	Not	Which Instead Means
Celsius	'Celsius'	'C'	coulomb
gram	'gram'	'g'	<standard free fall>
gallon	'gallon'	'gal'	<acceleration>
radian	'radian'	'rad'	<absorbed dose>
Newton	'newton' or 'N'	'nt'	nit (unit of photometry)

For additional information on this units library, please consult the manual pages that come with the distribution.

Appendix B File Format Specification

This appendix specifies the netCDF file format version 1. This format will be in use at least through netCDF library version 3.0.

The format is first presented formally, using a BNF grammar notation. In the grammar, optional components are enclosed between braces ('[' and ']'). Comments follow '//' characters. Nonterminals are in lower case, and terminals are in upper case. A sequence of zero or more occurrences of an entity are denoted by '[entity ...]'.

The Format in Detail

```
netcdf_file := header  data

header  := magic  numrecs  dim_array  gatt_array  var_array

magic   := 'C'  'D'  'F'  VERSION_BYTE

VERSION_BYTE := '\001'    // the file format version number

numrecs    := NON_NEG

dim_array  :=  ABSENT | NC_DIMENSION  nelems  [dim ...]

gatt_array :=  att_array  // global attributes

att_array  :=  ABSENT | NC_ATTRIBUTE  nelems  [attr ...]

var_array  :=  ABSENT | NC_VARIABLE   nelems  [var ...]

ABSENT  := ZERO  ZERO     // Means array not present (equivalent to
                          // nelems == 0).

nelems  := NON_NEG        // number of elements in following sequence

dim     := name  dim_size

name    := string

dim_size := NON_NEG       // If zero, this is the record dimension.
                          // There can be at most one record dimension.

attr    := name  nc_type  nelems  [values]

nc_type := NC_BYTE | NC_CHAR | NC_SHORT | NC_LONG | NC_FLOAT | NC_DOUBLE
```

```
var     := name  nelems  [dimid ...]  vatt_array  nc_type  vsize  begin
                          // nelems is the rank (dimensionality) of the
                          // variable; 0 for scalar, 1 for vector, 2 for
                          // matrix, ...

vatt_array :=  att_array  // variable-specific attributes

dimid   := NON_NEG        // Dimension ID (index into dim_array) for
                          // variable shape.  We say this is a "record
                          // variable" if and only if the first
                          // dimension is the record dimension.

vsize    := NON_NEG       // Variable size.  If not a record variable,
                          // the amount of space, in bytes, allocated to
                          // that variable's data.  This number is the
                          // product of the dimension sizes times the
                          // size of the type, padded to a four byte
                          // boundary.  If a record variable, it is the
                          // amount of space per record.  The netCDF
                          // "record size" is calculated as the sum of
                          // the vsize's of the record variables.

begin   := NON_NEG        // Variable start location.  The offset in
                          // bytes (seek index) in the file of the
                          // beginning of data for this variable.

data    := non_recs  recs

non_recs := [values ...]  // Data for first non-record var, second
                          // non-record var, ...

recs    := [rec ...]      // First record, second record, ...

rec     := [values ...]   // Data for first record variable for record
                          // n, second record variable for record n, ...
                          // See the note below for a special case.

values  := [bytes] | [chars] | [shorts] | [ints] | [floats] | [doubles]

string  := nelems  [chars]

bytes   := [BYTE ...]  padding

chars   := [CHAR ...]  padding

shorts  := [SHORT ...]  padding

ints    := [INT ...]
```

```
floats  := [FLOAT ...]

doubles := [DOUBLE ...]

padding := <0, 1, 2, or 3 bytes to next 4-byte boundary>
                         // In header, padding is with 0 bytes.  In
                         // data, padding is with variable's fill-value.

NON_NEG := <INT with non-negative value>

ZERO    := <INT with zero value>

BYTE    := <8-bit byte>

CHAR    := <8-bit ACSII/ISO encoded character>

SHORT   := <16-bit signed integer, Bigendian, two's complement>

INT     := <32-bit signed integer, Bigendian, two's complement>

FLOAT   := <32-bit IEEE single-precision float, Bigendian>

DOUBLE  := <64-bit IEEE double-precision float, Bigendian>

// tags are 32-bit INTs
NC_BYTE      := 1        // data is array of 8 bit signed integer
NC_CHAR      := 2        // data is array of characters, i.e., text
NC_SHORT     := 3        // data is array of 16 bit signed integer
NC_LONG      := 4        // data is array of 32 bit signed integer
NC_FLOAT     := 5        // data is array of IEEE single precision float
NC_DOUBLE    := 6        // data is array of IEEE double precision float
NC_DIMENSION := 10
NC_VARIABLE  := 11
NC_ATTRIBUTE := 12
```

Computing File Offsets

To calculate the offset (position within the file) of a specified data value, let *external_sizeof* be the external size in bytes of one data value of the appropriate type for the specified variable, *nc_type*:

```
NC_BYTE        1
NC_CHAR        1
NC_SHORT       2
NC_LONG        4
NC_FLOAT       4
NC_DOUBLE      8
```

On open() (or endef()), scan through the array of variables, denoted *var_array* above, and sum the *vsize* fields of "record" variables to compute *recsize*.

Form the the products of the dimension sizes for the variable from right to left, skipping the leftmost (record) dimension for record variables, and storing the results in a *product* array for each variable. For example:

```
Non-record variable:

        dimension sizes:        [  5  3  2 7]
        product:                [210 42 14 7]

Record variable:

        dimension sizes:        [0  2  9 4]
        product:                [0 72 36 4]
```

At this point, the left-most product, when rounded up to the next multiple of 4, is the variable size, *vsize*, in the grammar above. For example, in the non-record variable above, the value of the *vsize* field is 212 (210 rounded up to a multiple of 4). For the record variable, the value of *vsize* is just 72, since this is already a multiple of 4.

Let *coord* be an array of the coordinates of the desired data value, and *offset* be the desired result. Then *offset* is just the file offset of the first data value of the desired variable (its *begin* field) added to the inner product of the *coord* and *product* vectors times the size, in bytes, of each datum for the variable. Finally, if the variable is a record variable, the product of the record number, '`coord[0]`', and the record size, *recsize* is added to yield the final *offset* value.

In pseudo-C code, here's the calculation of *offset*:

```
for (innerProduct = i = 0; i < var.rank; i++)
        innerProduct += product[i] * coord[i]
offset = var.begin;
offset += external_sizeof * innerProduct
if(IS_RECVAR(var))
        offset += coord[0] * recsize;
```

So, to get the data value (in external representation):

```
lseek(fd, offset, SEEK_SET);
read(fd, buf, external_sizeof);
```

A special case: Where there is exactly one record variable, we drop the restriction that each record be four-byte aligned, so in this case there is no record padding.

Examples

By using the grammar above, we can derive the smallest valid netCDF file, having no dimensions, no variables, no attributes, and hence, no data. A CDL representation of the empty netCDF file is

```
netcdf empty { }
```

This empty netCDF file has 32 bytes, as you may verify by using '`ncgen -b empty.cdl`' to generate it from the CDL representation. It begins with the four-byte "magic number" that identifies it as a netCDF version 1 file: 'C', 'D', 'F', '\001'. Following are seven 32-bit integer zeros representing the number of records, an empty array of dimensions, an empty array of global attributes, and an empty array of variables.

Below is an (edited) dump of the file produced on a big-endian machine using the Unix command

```
od -xcs empty.nc
```

Each 16-byte portion of the file is displayed with 4 lines. The first line displays the bytes in hexadecimal. The second line displays the bytes as characters. The third line displays each group of two bytes interpreted as a signed 16-bit integer. The fourth line (added by human) presents the interpretation of the bytes in terms of netCDF components and values.

```
  4344    4601    0000    0000    0000    0000    0000    0000
 C   D   F 001  \0  \0  \0  \0  \0  \0  \0  \0  \0  \0  \0  \0
 17220   17921   00000   00000   00000   00000   00000   00000
[magic number ] [  0 records   ] [  0 dimensions   (ABSENT)    ]

  0000    0000    0000    0000    0000    0000    0000    0000
 \0  \0  \0  \0  \0  \0  \0  \0  \0  \0  \0  \0  \0  \0  \0  \0
 00000   00000   00000   00000   00000   00000   00000   00000
[  0 global atts  (ABSENT)    ] [  0 variables    (ABSENT)    ]
```

As a slightly less trivial example, consider the CDL

```
netcdf tiny {
dimensions:
        dim = 5;
variables:
        short vx(dim);
data:
        vx = 3, 1, 4, 1, 5 ;
}
```

which corresponds to a 92-byte netCDF file. The following is an edited dump of this file:

```
  4344    4601    0000    0000    0000    000a    0000    0001
 C   D   F 001  \0  \0  \0  \0  \0  \0  \0  \n  \0  \0  \0 001
 17220   17921   00000   00000   00000   00010   00000   00001
[magic number ] [  0 records   ] [NC_DIMENSION ] [ 1 dimension ]

  0000    0003    6469    6d00    0000    0005    0000    0000
 \0  \0  \0 003   d   i   m  \0  \0  \0  \0 005  \0  \0  \0  \0
 00000   00003   25705   27904   00000   00005   00000   00000
[  3 char name = "dim"        ] [ size = 5    ] [ 0 global atts

  0000    0000    0000    000b    0000    0001    0000    0002
  \0  \0  \0  \0  \0  \0  \0 013  \0  \0  \0 001  \0  \0  \0 002
   00000   00000   00000   00011   00000   00001   00000   00002
  (ABSENT)     ] [NC_VARIABLE  ] [ 1 variable  ] [ 2 char name =

     7678    0000    0000    0001    0000    0000    0000    0000
    v   x  \0  \0  \0  \0  \0 001  \0  \0  \0  \0  \0  \0  \0  \0
   30328   00000   00000   00001   00000   00000   00000   00000
  "vx"         ] [1 dimension  ] [ with ID 0   ] [ 0 attributes

     0000    0000    0000    0003    0000    000c    0000    0050
  \0  \0  \0  \0  \0  \0  \0 003  \0  \0  \0  \f  \0  \0  \0   P
   00000   00000   00000   00003   00000   00012   00000   00080
  (ABSENT)     ] [type NC_SHORT] [size 12 bytes] [offset:    80]

     0003    0001    0004    0001    0005    8001
  \0 003  \0 001  \0 004  \0 001  \0 005 200 001
   00003   00001   00004   00001   00005  -32767
 [    3] [    1] [    4] [    1] [    5] [fill ]
```

Appendix C Summary of C Interface

```
int nccreate (const char* filename, int cmode);
int ncopen (const char* filename, int mode);
int ncredef (int ncid);
int ncendef (int ncid);
int ncclose (int ncid);
int ncinquire (int ncid, int* ndims, int* nvars, int* natts, int* recdim);
int ncsync (int ncid);
int ncabort (int ncid);
int ncdimdef (int ncid, const char* name, long length);
int ncdimid (int ncid, const char* name);
int ncdiminq (int ncid, int dimid, char* name, long* length);
int ncdimrename (int ncid, int dimid, const char* name);
int ncvardef (int ncid, const char* name, nc_type datatype, int ndims,
              const int dimids[]);
int ncvarid (int ncid, const char* name);
int ncvarinq (int ncid, int varid, char* name, nc_type* datatype, int* ndims,
              int dimids[], int* natts);
int ncvarput1 (int ncid, int varid, const long coords[], const void* value);
int ncvarget1 (int ncid, int varid, const long coords[], void* value);
int ncvarput (int ncid, int varid, const long start[], const long count[],
              const void* value);
int ncvarget (int ncid, int varid, const long start[], const long count[],
              void* value);
int ncvarputg (int ncid, int varid, const long start[], const long count[],
              const long stride[], const long imap[], const void* value);
int ncvargetg (int ncid, int varid, const long start[], const long count[],
              const long stride[], const long imap[], void* value);
int ncvarrename (int ncid, int varid, const char* name);
int ncattput (int ncid, int varid, const char* name, nc_type datatype, int len,
              const void* value);
int ncattinq (int ncid, int varid, const char* name, nc_type* datatype,
              int* len);
int ncattget (int ncid, int varid, const char* name, void* value);
int ncattcopy (int incdf, int invar, const char* name, int outcdf, int outvar);
int ncattname (int ncid, int varid, int attnum, char* name);
int ncattrename (int ncid, int varid, const char* name, const char* newname);
int ncattdel (int ncid, int varid, const char* name);
int nctypelen (nc_type datatype);
int ncsetfill (int ncid, int fillmode);
int ncrecput(int ncid, long recnum, void* const datap[]);
int ncrecget(int ncid, long recnum, void* datap[]);
int ncrecinq(int ncid, int *nrecvars, int recvarids[], long recsizes[]);
```

Appendix D Summary of FORTRAN Interface

Input parameters are in upper case, output parameters are in lower case. The FORTRAN types of all the parameters are listed alphabetically by parameter name below the subroutine and function declarations.

```
INTEGER FUNCTION NCCRE (FILENAME,CLOBMODE, rcode)
INTEGER FUNCTION NCOPN (FILENAME,RWMODE, rcode)
SUBROUTINE NCREDF (NCID, rcode)
SUBROUTINE NCENDF (NCID, rcode)
SUBROUTINE NCCLOS (NCID, rcode)
SUBROUTINE NCINQ (NCID, ndims,nvars,natts,recdim,rcode)
SUBROUTINE NCSNC (NCID, rcode)
SUBROUTINE NCABOR (NCID, rcode)
INTEGER FUNCTION NCDDEF (NCID,DIMNAME,SIZE, rcode)
INTEGER FUNCTION NCDID (NCID,DIMNAME, rcode)
SUBROUTINE NCDINQ (NCID,DIMID, dimname,size,rcode)
SUBROUTINE NCDREN (NCID,DIMID,DIMNAME, rcode)
INTEGER FUNCTION NCVDEF (NCID,VARNAME,DATATYPE,NVDIMS,VDIMS, rcode)
INTEGER FUNCTION NCVID (NCID,VARNAME, rcode)
SUBROUTINE NCVINQ (NCID,VARID, varname,datatype,nvdims,vdims,nvatts,rcode)
SUBROUTINE NCVPT1 (NCID,VARID,INDICES,VALUE, rcode)
SUBROUTINE NCVP1C (NCID,VARID,INDICES, CHVAL, rcode)
SUBROUTINE NCVGT1 (NCID,VARID,INDICES, value, rcode)
SUBROUTINE NCVG1C (NCID,VARID,INDICES, chval, rcode)
SUBROUTINE NCVPT (NCID,VARID,START,COUNTS,VALUE, rcode)
SUBROUTINE NCVPTC (NCID,VARID,START,COUNTS,STRING,LENSTR, rcode)
SUBROUTINE NCVPTG (NCID,VARID,START,COUNTS,STRIDE,IMAP,VALUE, rcode)
SUBROUTINE NCVPGC (NCID,VARID,START,COUNTS,STRIDE,IMAP,STRING,rcode)
SUBROUTINE NCVGT (NCID,VARID,START,COUNTS, value,rcode)
SUBROUTINE NCVGTC (NCID,VARID,START,COUNTS, string,LENSTR,rcode)
SUBROUTINE NCVGTG (NCID,VARID,START,COUNTS,STRIDE,IMAP,value,rcode)
SUBROUTINE NCVGGC (NCID,VARID,START,COUNTS,STRIDE,IMAP,string,rcode)
SUBROUTINE NCVREN (NCID,VARID,VARNAME, rcode)
SUBROUTINE NCAPT (NCID,VARID,ATTNAME,DATATYPE,ATTLEN,VALUE, rcode)
SUBROUTINE NCAPTC (NCID,VARID,ATTNAME,DATATYPE,LENSTR,STRING, rcode)
SUBROUTINE NCAINQ (NCID,VARID,ATTNAME, datatype,attlen,rcode)
SUBROUTINE NCAGT (NCID,VARID,ATTNAME, value,rcode)
SUBROUTINE NCAGTC (NCID,VARID,ATTNAME, string,LENSTR,rcode)
SUBROUTINE NCACPY (INNCID,INVARID,ATTNAME,OUTNCID,OUTVARID, rcode)
SUBROUTINE NCANAM (NCID,VARID,ATTNUM, attname,rcode)
SUBROUTINE NCAREN (NCID,VARID,ATTNAME,NEWNAME, rcode)
SUBROUTINE NCADEL (NCID,VARID,ATTNAME, rcode)
INTEGER FUNCTION NCTLEN (DATATYPE, rcode)
SUBROUTINE NCPOPT (NCOPTS)
SUBROUTINE NCGOPT (ncopts)
INTEGER FUNCTION NCSFIL (NCID,FILLMODE, rcode)
INTEGER ATTLEN          ! number of elements in an attribute vector
CHARACTER*(*) ATTNAME   ! attribute name
INTEGER ATTNUM          ! attribute number
CHARACTER CHVAL         ! character value of variable or attribute
INTEGER CLOBMODE        ! NCCLOB or NCNOCLOB
INTEGER COUNTS(NVDIMS)  ! edge lengths of block of values
INTEGER DATATYPE        ! type: NCBYTE, ..., or NCDOUBLE
INTEGER DIMID           ! dimension ID
CHARACTER*(*) DIMNAME   ! dimension name
CHARACTER*(*) FILENAME  ! name of netCDF file
INTEGER FILLMODE        ! NCNOFILL or NCFILL, for setting fill mode
INTEGER IMAP(NVDIMS)    ! index mapping vector
INTEGER INDICES(NDIMS)  ! coordinates of a single element of a variable
INTEGER INNCID          ! input netCDF ID
INTEGER INVARID         ! input variable ID
INTEGER LENSTR          ! length of character array value
INTEGER NATTS           ! number of global attributes
INTEGER NCID            ! netCDF ID, returned by NCCRE or NCOPN
INTEGER NCOPTS          ! error-handling option flag
INTEGER NDIMS           ! number of dimensions
CHARACTER*(*) NEWNAME   ! new attribute name
INTEGER NVARS           ! number of variables
INTEGER NVATTS          ! number of attributes assigned to a variable
INTEGER NVDIMS          ! number of dimensions in a variable
INTEGER OUTNCID         ! output netCDF ID
INTEGER OUTVARID        ! output variable ID
INTEGER RCODE           ! returned error code, 0 if no errors
INTEGER RECDIM          ! dimension ID of unlimited dimension
INTEGER RWMODE          ! NCWRITE or NCNOWRIT
INTEGER SIZE            ! dimension size
INTEGER START(NVDIMS)   ! index of first value to be accessed
INTEGER STRIDE(NVDIMS)  ! netCDF variable dimensional strides
CHARACTER*(*) STRING    ! character array value of variable or attribute
DOUBLE VALUE            ! double precision value of variable or attribute
REAL VALUE              ! real value of variable or attribute
INTEGER VALUE           ! integer value of variable or attribute
INTEGER VARID           ! variable ID from NCVDEF or NCVID, or NCGLOBAL
CHARACTER*(*) VARNAME   ! variable name
INTEGER VDIMS(NDIMS)    ! dimension IDs for a variable, giving its shape
```

Function and Variable Index

General Index

Table of Contents